

Un exemple de client / serveur AngularJS / Spring 4

serge.tahe at univ-angers.fr, juillet 2014

Table des matières

1 INTRODUCTION.....	6
1.1 L'ARCHITECTURE DE L'APPLICATION.....	6
1.2 LES OUTILS UTILISÉS.....	7
1.3 LES FONCTIONNALITÉS DE L'APPLICATION.....	7
1.3.1 CRÉATION DE LA BASE DE DONNÉES.....	8
1.3.2 MISE EN OEUVRE DU SERVEUR WEB / JSON.....	9
1.3.3 MISE EN OEUVRE DU CLIENT ANGULAR.....	11
2 LE SERVEUR SPRING 4.....	17
2.1 LA BASE DE DONNÉES.....	17
2.1.1 LA TABLE [MEDECINS].....	18
2.1.2 LA TABLE [CLIENTS].....	19
2.1.3 LA TABLE [CRENEAUX].....	19
2.1.4 LA TABLE [RV].....	20
2.2 INTRODUCTION À SPRING DATA.....	20
2.2.1 LA CONFIGURATION MAVEN DU PROJET.....	21
2.2.2 LA COUCHE [JPA].....	23
2.2.3 LA COUCHE [DAO].....	24
2.2.4 LA COUCHE [CONSOLE].....	26
2.2.5 CONFIGURATION MANUELLE DU PROJET SPRING DATA.....	29
2.2.6 CRÉATION D'UNE ARCHIVE EXÉCUTABLE.....	33
2.2.7 CRÉER UN NOUVEAU PROJET SPRING DATA.....	35
2.3 LE PROJET ECLIPSE DU SERVEUR.....	38
2.4 LA CONFIGURATION MAVEN.....	38
2.5 LES ENTITÉS JPA.....	41
2.6 LA COUCHE [DAO].....	47
2.7 LA COUCHE [MÉTIER].....	49
2.7.1 LES ENTITÉS.....	49
2.7.2 LE SERVICE.....	50
2.8 LA CONFIGURATION DU PROJET.....	53
2.9 LES TESTS DE LA COUCHE [MÉTIER].....	54
2.10 LE PROGRAMME CONSOLE.....	57
2.11 INTRODUCTION À SPRING MVC.....	59
2.11.1 LE PROJET DE DÉMONSTRATION.....	59
2.11.2 CONFIGURATION MAVEN.....	60
2.11.3 L'ARCHITECTURE D'UN SERVICE SPRING REST.....	61
2.11.4 LE CONTRÔLEUR C.....	62
2.11.5 LE MODÈLE M.....	63
2.11.6 CONFIGURATION DU PROJET.....	63
2.11.7 EXÉCUTION DU PROJET.....	64
2.11.8 CRÉATION D'UNE ARCHIVE EXÉCUTABLE.....	66
2.11.9 DÉPLOYER L'APPLICATION SUR UN SERVEUR TOMCAT.....	68
2.11.10 CRÉER UN NOUVEAU PROJET WEB.....	70
2.12 LA COUCHE [WEB].....	71
2.12.1 CONFIGURATION MAVEN.....	71
2.12.2 L'INTERFACE DU SERVICE WEB.....	72
2.12.3 LE SQUELETTE DU CONTRÔLEUR [RdvMEDECINSCONTROLLER].....	78
2.12.4 LES MODÈLES DU SERVICE WEB.....	80
2.12.5 LA CLASSE STATIC.....	82
2.12.6 LA MÉTHODE [INIT] DU CONTRÔLEUR.....	83
2.12.7 L'URL [/GETALLMEDECINS].....	83
2.12.8 L'URL [/GETALLCLIENTS].....	84
2.12.9 L'URL [/GETALLCRENEAUX/{IDMEDECIN}].....	85
2.12.10 L'URL [/GETRvMEDECINJOUR/{IDMEDECIN}/{JOUR}].....	88
2.12.11 L'URL [/GETAGENDAMEDECINJOUR/{IDMEDECIN}/{JOUR}].....	90
2.12.12 L'URL [/GETMEDECINById/{ID}].....	92
2.12.13 L'URL [/GETCLIENTById/{ID}].....	93
2.12.14 L'URL [/GETCRENEAUById/{ID}].....	94
2.12.15 L'URL [/GETRvById/{ID}].....	95
2.12.16 L'URL [/AJOUTERRv].....	96
2.12.17 L'URL [/SUPPRIMERRv].....	99
2.12.18 CONFIGURATION DU SERVICE WEB.....	101

<u>2.12.19</u> LA CLASSE EXÉCUTABLE DU SERVICE WEB.....	102
<u>2.13</u> INTRODUCTION À SPRING SECURITY.....	104
<u>2.13.1</u> CONFIGURATION MAVEN.....	105
<u>2.13.2</u> LES VUES THYMELEAF	106
<u>2.13.3</u> CONFIGURATION SPRING MVC.....	109
<u>2.13.4</u> CONFIGURATION SPRING SECURITY.....	110
<u>2.13.5</u> CLASSE EXÉCUTABLE.....	111
<u>2.13.6</u> TESTS DE L'APPLICATION.....	111
<u>2.13.7</u> CONCLUSION.....	113
<u>2.14</u> MISE EN PLACE DE LA SÉCURITÉ SUR LE SERVICE WEB DES RENDEZ-VOUS.....	114
<u>2.14.1</u> LA BASE DE DONNÉES.....	114
<u>2.14.2</u> LE NOUVEAU PROJET ECLIPSE DU [MÉTIER, DAO, JPA].....	115
<u>2.14.3</u> LES NOUVELLES ENTITÉS [JPA].....	116
<u>2.14.4</u> MODIFICATIONS DE LA COUCHE [DAO].....	117
<u>2.14.5</u> LES CLASSES DE GESTION DES UTILISATEURS ET DES RÔLES.....	119
<u>2.14.6</u> TESTS DE LA COUCHE [DAO].....	121
<u>2.14.7</u> CONCLUSION INTERMÉDIAIRE.....	125
<u>2.14.8</u> LE PROJET ECLIPSE DE LA COUCHE [WEB].....	125
<u>2.14.9</u> TESTS DU SERVICE WEB.....	128
<u>2.15</u> CONCLUSION.....	131
<u>3</u> LE CLIENT ANGULAR JS.....	133
<u>3.1</u> RÉFÉRENCES DU FRAMEWORK ANGULAR JS.....	133
<u>3.2</u> ARCHITECTURE DU CLIENT ANGULAR.....	133
<u>3.3</u> LES VUES DU CLIENT ANGULAR.....	136
<u>3.4</u> CONFIGURATION DU PROJET ANGULAR.....	140
<u>3.5</u> LA PAGE INITIALE DU CLIENT ANGULAR.....	144
<u>3.6</u> DÉCOUVERTE DE BOOTSTRAP.....	147
<u>3.6.1</u> EXEMPLE 1.....	147
<u>3.6.2</u> EXEMPLE 2.....	147
<u>3.6.3</u> EXEMPLE 3.....	148
<u>3.6.4</u> EXEMPLE 4.....	149
<u>3.6.5</u> EXEMPLE 5.....	150
<u>3.6.6</u> EXEMPLE 6.....	151
<u>3.6.7</u> EXEMPLE 7.....	152
<u>3.6.8</u> EXEMPLE 8.....	154
<u>3.6.9</u> EXEMPLE 9.....	155
<u>3.6.10</u> CONCLUSION.....	157
<u>3.7</u> DÉCOUVERTE D'ANGULAR JS.....	157
<u>3.7.1</u> EXEMPLE 1 : LE MODÈLE MVC D'ANGULAR.....	158
<u>3.7.2</u> EXEMPLE 2 : LOCALISATION DES DATES.....	161
<u>3.7.3</u> EXEMPLE 3 : INTERNATIONALISATION DES TEXTES.....	167
<u>3.7.4</u> EXEMPLE 4 : UN SERVICE DE CONFIGURATION.....	173
<u>3.7.5</u> EXEMPLE 5 : PROGRAMMATION ASYNCHRONE.....	177
<u>3.7.6</u> EXEMPLE 6 : LES SERVICES HTTP.....	184
<u>3.7.6.1</u> La vue V.....	184
<u>3.7.6.2</u> Le contrôleur C et le modèle M.....	186
<u>3.7.6.3</u> Le contrôleur C.....	187
<u>3.7.6.4</u> Le service [dao].....	191
<u>3.7.6.5</u> Tests de l'application - 1.....	195
<u>3.7.6.6</u> Modification du serveur web / JSON.....	198
<u>3.7.6.7</u> Tests de l'application - 2.....	200
<u>3.7.7</u> EXEMPLE 7 : LISTE DES CLIENTS.....	203
<u>3.7.7.1</u> La vue V.....	203
<u>3.7.7.2</u> Le contrôleur C et le modèle M.....	204
<u>3.7.7.3</u> Modification du service web - 1.....	206
<u>3.7.7.4</u> Tests de l'application – 1.....	207
<u>3.7.7.5</u> Modification du service web – 2.....	209
<u>3.7.7.6</u> Tests de l'application – 2.....	210
<u>3.7.7.7</u> Utilisation d'une directive.....	213
<u>3.7.7.8</u> Tests de l'application – 3.....	215
<u>3.7.8</u> EXEMPLE 8 : L'AGENDA D'UN MÉDECIN.....	216
<u>3.7.8.1</u> La vue V de l'application.....	216
<u>3.7.8.2</u> Le formulaire.....	217

<u>3.7.8.3</u> Le contrôleur C.....	218
<u>3.7.8.4</u> Affichage de l'agenda.....	222
<u>3.7.8.5</u> Modification du serveur web.....	223
<u>3.7.8.6</u> Utilisation de directives.....	224
<u>3.7.9</u> EXEMPLE 9 : CRÉER ET ANNULER DES RÉSERVATIONS.....	225
<u>3.7.9.1</u> La vue V de l'application.....	225
<u>3.7.9.2</u> Le contrôleur C.....	227
<u>3.7.9.3</u> Initialisation du contrôleur C.....	227
<u>3.7.9.4</u> Obtention de l'agenda.....	229
<u>3.7.9.5</u> Réservation d'un créneau horaire.....	229
<u>3.7.9.6</u> Modification serveur.....	231
<u>3.7.9.7</u> Tests.....	232
<u>3.7.9.8</u> Suppression d'un rendez-vous.....	234
<u>3.7.9.9</u> Modification serveur.....	235
<u>3.7.10</u> EXEMPLE 10 : CRÉER ET ANNULER DES RÉSERVATIONS - 2.....	236
<u>3.7.10.1</u> La vue V de l'application.....	236
<u>3.7.10.2</u> Le contrôleur C.....	237
<u>3.7.11</u> EXEMPLE 11 : UNE DIRECTIVE [SELECTENABLE2].....	239
<u>3.7.11.1</u> La vue V.....	239
<u>3.7.11.2</u> Le code HTML de la vue.....	239
<u>3.7.11.3</u> La directive [selectEnable2].....	240
<u>3.7.11.4</u> Le contrôleur C.....	240
<u>3.7.11.5</u> Les tests.....	241
<u>3.7.12</u> EXEMPLE 12 : UNE DIRECTIVE [LIST].....	241
<u>3.7.12.1</u> La directive [list].....	242
<u>3.7.12.2</u> Le code HTML.....	243
<u>3.7.12.3</u> Le contrôleur C.....	244
<u>3.7.12.4</u> Les tests.....	244
<u>3.7.13</u> EXEMPLE 13 : MISE À JOUR DU MODÈLE D'UNE DIRECTIVE.....	244
<u>3.7.13.1</u> Les vues V.....	244
<u>3.7.13.2</u> La page HTML.....	245
<u>3.7.13.3</u> La directive [list2].....	246
<u>3.7.13.4</u> Le contrôleur C.....	246
<u>3.7.14</u> EXEMPLE 14 : LES DIRECTIVES [WAITING] ET [ERRORS].....	250
<u>3.7.14.1</u> Le nouveau code HTML.....	250
<u>3.7.14.2</u> La directive [waiting].....	251
<u>3.7.14.3</u> La directive [errors].....	251
<u>3.7.15</u> EXEMPLE 15 : NAVIGATION.....	252
<u>3.7.15.1</u> Les vues V de l'application.....	252
<u>3.7.15.2</u> Organisation du code.....	253
<u>3.7.15.3</u> Le conteneur des vues.....	253
<u>3.7.15.4</u> Le module de l'application.....	254
<u>3.7.15.5</u> Le contrôleur du conteneur de vues.....	254
<u>3.7.15.6</u> La barre de navigation.....	255
<u>3.7.15.7</u> La vue [/page1] et son contrôleur.....	257
<u>3.7.15.8</u> Contrôle de la navigation.....	257
<u>3.7.16</u> CONCLUSION.....	258
3.8 LE CLIENT FINAL ANGULAR.....	259
<u>3.8.1</u> STRUCTURE DU PROJET.....	259
<u>3.8.2</u> LES DÉPENDANCES DU PROJET.....	259
<u>3.8.3</u> LA PAGE MAÎTRE [APP.HTML].....	260
<u>3.8.4</u> LES VUES DE L'APPLICATION.....	262
<u>3.8.5</u> FONCTIONNALITÉS DE L'APPLICATION.....	264
<u>3.8.6</u> LE MODULE [MAIN.JS].....	269
<u>3.8.7</u> LE CONTRÔLEUR DE LA PAGE MAÎTRE.....	270
<u>3.8.8</u> GESTION DE LA TÂCHE ASYNCHRONE.....	272
<u>3.8.9</u> CONTRÔLE DE LA NAVIGATION.....	273
<u>3.8.10</u> LES SERVICES.....	276
<u>3.8.11</u> LES DIRECTIVES.....	278
<u>3.8.12</u> LE CONTRÔLEUR [LOGINCTRL].....	281
<u>3.8.13</u> LE CONTRÔLEUR [HOMECTRL].....	284
<u>3.8.14</u> LE CONTRÔLEUR [AGENDACTRL].....	286
<u>3.8.15</u> LE CONTRÔLEUR [RESACTRL].....	288

<u>3.8.16</u> LA GESTION DES LANGUES.....	290
4 EXPLOITATION DE L'APPLICATION.....	291
<u>4.1</u> DÉPLOIEMENT DU SERVICE WEB SUR UN SERVEUR TOMCAT.....	291
<u>4.2</u> DÉPLOIEMENT DU CLIENT ANGULAR SUR LE SERVEUR TOMCAT.....	296
<u>4.3</u> LES ENTÈTES CORS.....	299
<u>4.4</u> DÉPLOIEMENT DU CLIENT ANGULAR SUR UNE TABLETTE ANDROID.....	300
<u>4.5</u> DÉPLOIEMENT DU CLIENT ANGULAR SUR L'ÉMULATEUR D'UN SMARTPHONE ANDROID.....	304
<u>5 CONCLUSION.....</u>	<u>306</u>
6 ANNEXES.....	308
<u>6.1</u> INSTALLATION DE STS (SPRING TOOL SUITE).....	308
<u>6.2</u> INSTALLATION DE [WAMP SERVER].....	309
<u>6.3</u> INSTALLATION DE [WEBSTORM].....	310
<u>6.3.1</u> INSTALLATION DE [NODE.JS].....	311
<u>6.3.2</u> INSTALLATION DE L'OUTIL [BOWER].....	311
<u>6.3.3</u> INSTALLATION DE [GIT].....	311
<u>6.3.4</u> CONFIGURATION DE [WEBSTORM].....	312
<u>6.4</u> INSTALLATION D'UN ÉMULATEUR POUR ANDROID.....	313
<u>6.5</u> INSTALLATION DU PLUGIN CHROME [ADVANCED REST CLIENT].....	314

1 Introduction

Nous nous proposons ici d'introduire deux frameworks à l'aide d'un exemple de client / serveur :

- **AngularJS** utilisé pour le client. Pour simplifier, il sera noté **Angular** par la suite ;
- **Spring 4** utilisé pour le serveur. Pour simplifier, il sera noté **Spring** par la suite ;

La compréhension de ce document nécessite certains pré-requis :

- un niveau intermédiaire en Java EE ;
- la connaissance de JPA (Java Persistence API) qui sera utilisé pour accéder à une base de données ;
- la connaissance d'au moins une version précédente de Spring pour connaître la philosophie de ce framework ;
- l'utilisation de Maven pour configurer des projets Java ;
- une connaissance de base des échanges HTTP dans une application web ;
- les balises courantes du langage HTML ;
- une connaissance basique du langage Javascript ;

Les autres connaissances nécessaires seront introduites et expliquées au fur et à mesure de l'étude de cas.

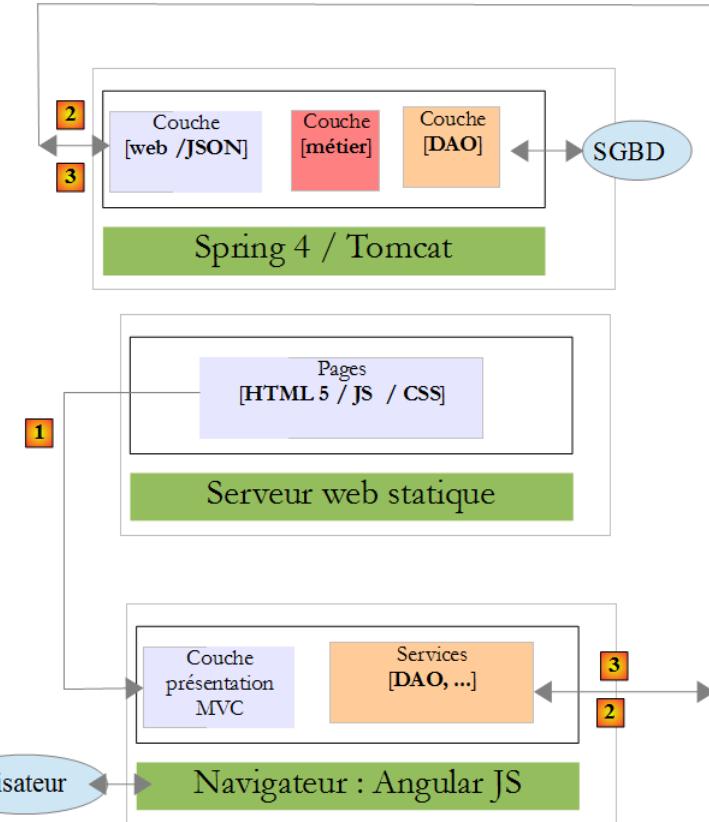
Ce document n'est pas un cours et il est incomplet à bien des égards. Pour approfondir les deux frameworks, on pourra utiliser les références suivantes :

- [ref1] : le livre "**Pro AngularJS**" écrit par **Adam Freeman** aux éditions **Apress**. C'est un excellent livre. Les codes source des exemples de ce livre sont disponibles gratuitement à l'URL [\[http://www.apress.com/downloadable/download/sample/sample_id/1527/\]](http://www.apress.com/downloadable/download/sample/sample_id/1527/) ;
- [ref2] : la documentation officielle d'Angular JS [\[https://docs.angularjs.org/guide\]](https://docs.angularjs.org/guide)
- [ref3] : le livre "**Spring Data**" chez o'Reilly [\[http://shop.oreilly.com/product/0636920024767.do\]](http://shop.oreilly.com/product/0636920024767.do) qui présente l'utilisation du framework [Spring Data] pour l'accès aux données que ce soit des bases de données relationnelles ou pas (NoSQL) ;
- [ref4] : le livre "**Pro Spring3**" aux éditions **Apress**. C'est la version précédente de Spring 4 mais les principaux concepts sont déjà là ;
- [ref5] : la documentation de référence de Spring 4 [\[http://docs.spring.io/spring/docs/current/spring-framework-reference/pdf/spring-framework-reference.pdf\]](http://docs.spring.io/spring/docs/current/spring-framework-reference/pdf/spring-framework-reference.pdf).

Les sources qui ont nourri ce document sont celles citées ci-dessus plus l'indispensable [\[http://stackoverflow.com/\]](http://stackoverflow.com/) pour les très nombreuses séances de débogage.

1.1 L'architecture de l'application

L'application étudiée aura l'architecture suivante :



- en [1], un serveur web délivre des pages statiques à un navigateur. Ces pages contiennent une application AngularJS construite sur le modèle MVC (Modèle – Vue – Contrôleur). Le modèle ici est à la fois celui des vues et celui du domaine représenté ici par la couche [Services] ;
- l'utilisateur va interagir avec les vues qui lui sont présentées dans le navigateur. Ses actions vont parfois nécessiter l'interrogation du serveur Spring 4 [2]. Celui-ci traitera la demande et rendra une réponse JSON (JavaScript Object Notation) [3]. Celle-ci sera utilisée pour mettre à jour la vue présentée à l'utilisateur.

1.2 Les outils utilisés

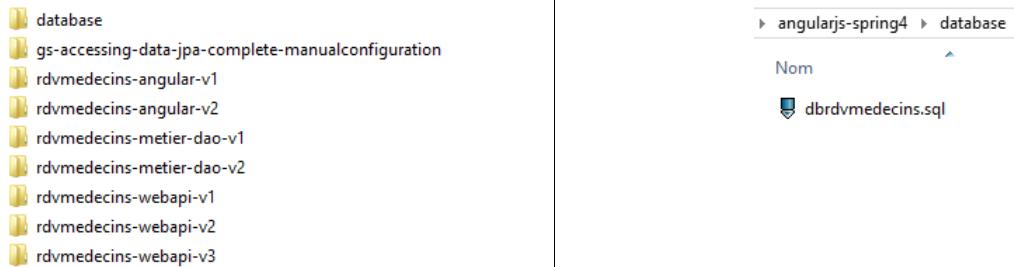
Dans ce document les outils de développement utilisés sont les suivants :

- [Spring Tool Suite](#) pour le serveur Spring : téléchargeable gratuitement ;
- [Webstorm](#) pour le client Angular : une version d'évaluation pour un mois est téléchargeable gratuitement ;
- [Wampserver](#) pour la gestion de la base de données MySQL 5 : téléchargeable gratuitement ;

L'installation de ces outils et d'autres est décrite au paragraphe 6, page 308.

1.3 Les fonctionnalités de l'application

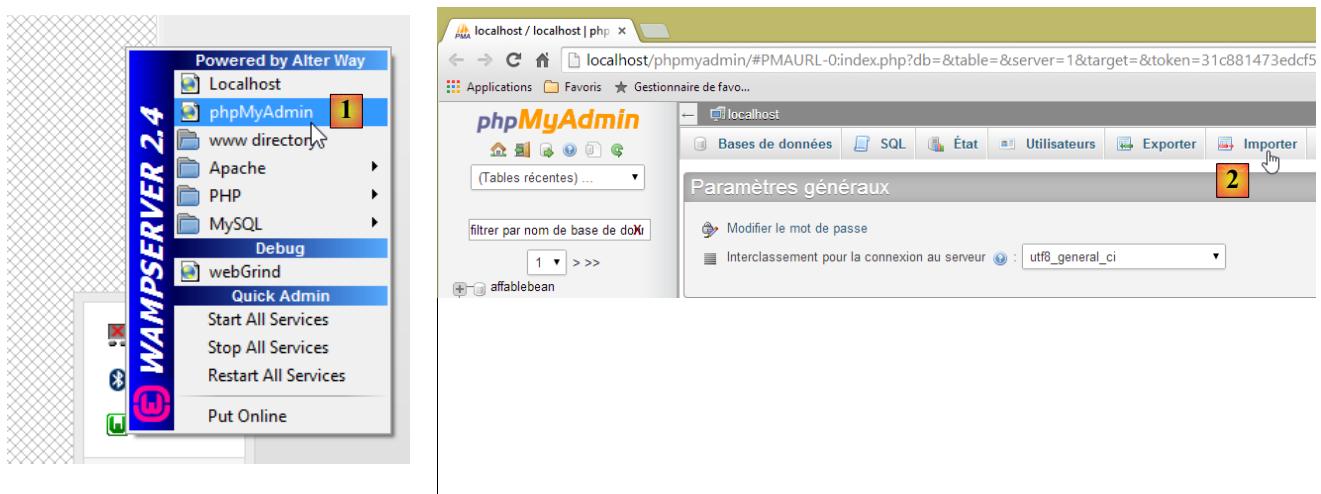
Le code de l'exemple est disponible à l'URL [<http://tahe.developpez.com/java/angularjs-spring4>] sous la forme d'un fichier *.zip* à télécharger.



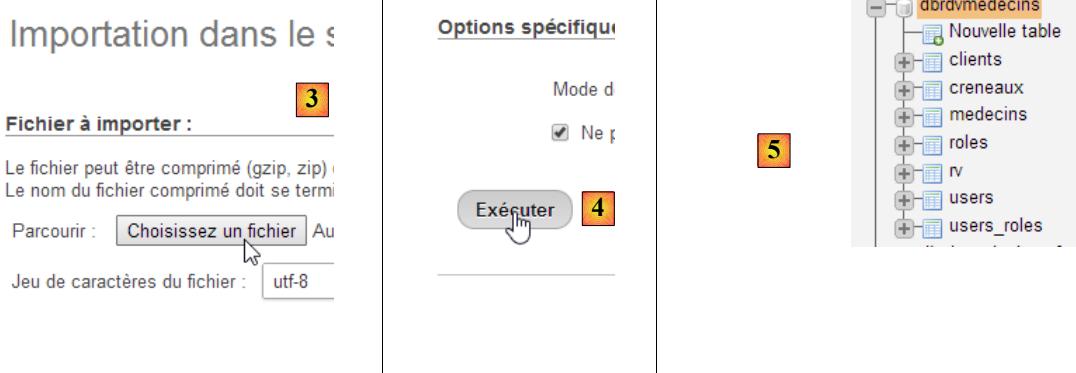
- le serveur est contenu dans les dossiers [rdvmedecins-metier-dao-v2] et [rdvmedecins-webapi-v3] ;
- le client est contenu dans le dossier [rdvmedecins-angular-v2] ;
- le script SQL de génération de la base de données MySQL5 est dans le dossier [database] ;

1.3.1 Création de la base de données

Pour tester l'application, nous créons d'abord la base de données avec le script SQL [dbrdvmedecins.sql]. Nous utilisons l'outil [PhpMyAdmin] de WampServer :



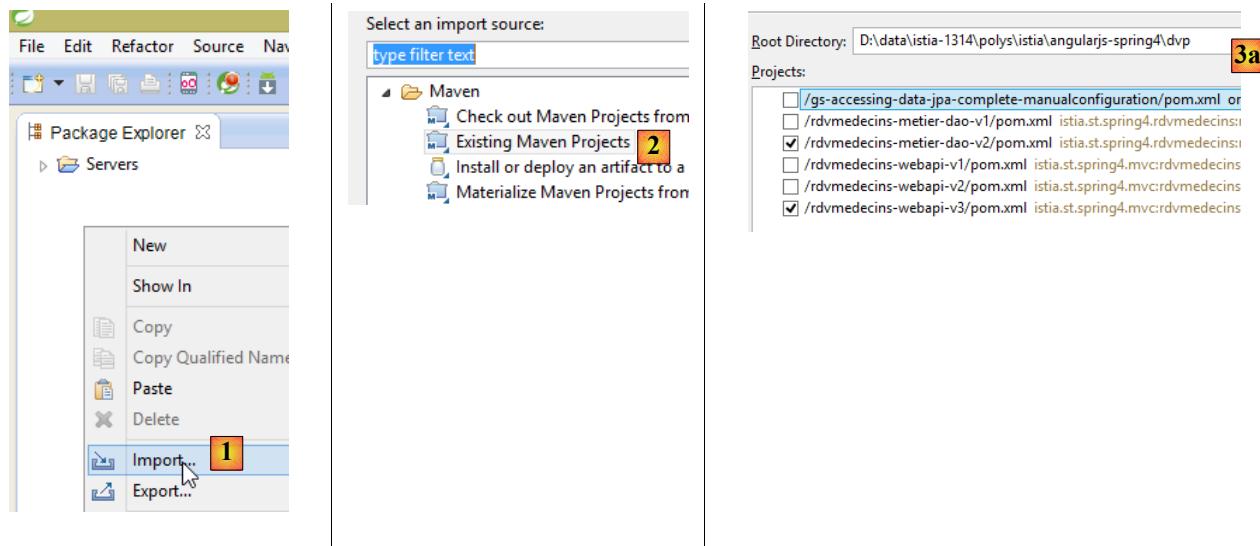
- en [1], on sélectionne l'outil [phpMyAdmin] de WampServer ;
- en [2], on choisit l'option [Importer] ;



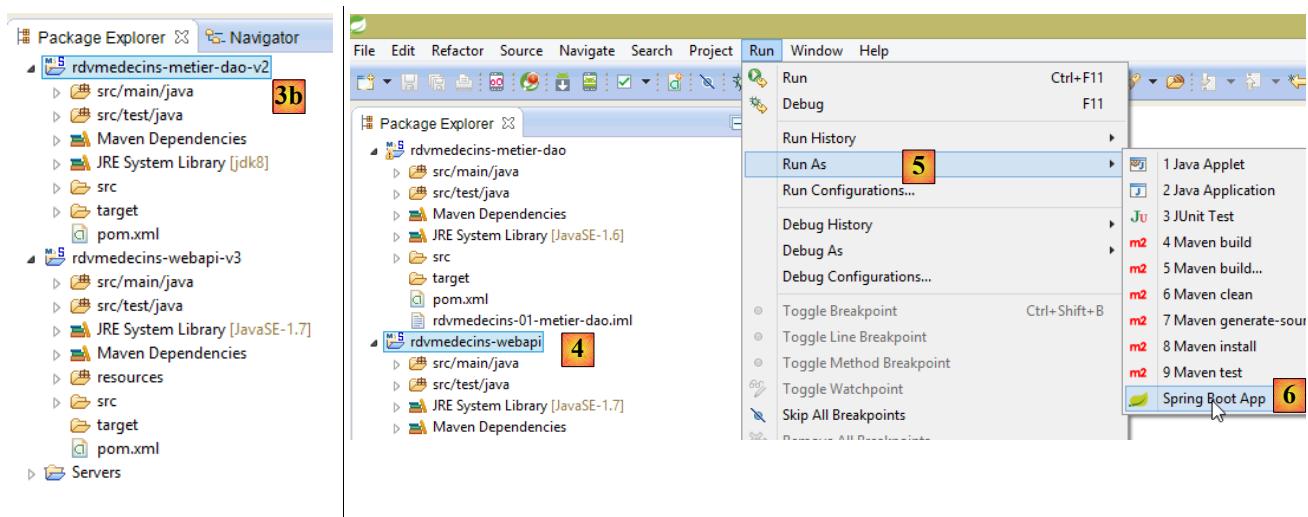
- en [3], on sélectionne le fichier [database/dbrdvmedecins.sql] ;
- en [4], on l'exécute ;
- en [5], la base de données créée.

1.3.2 Mise en oeuvre du serveur web / JSON

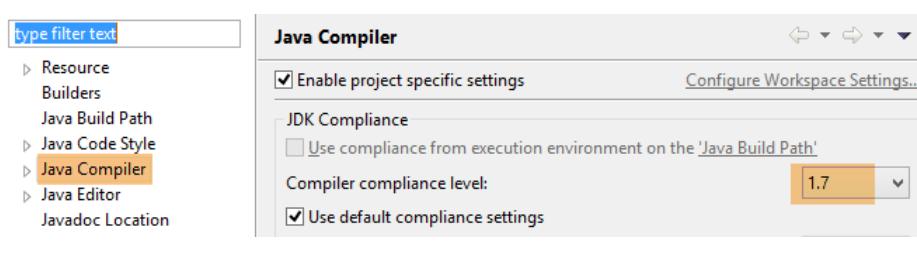
Avec Spring Tool Suite (STS), on importe les deux projets Maven du serveur Spring 4 :



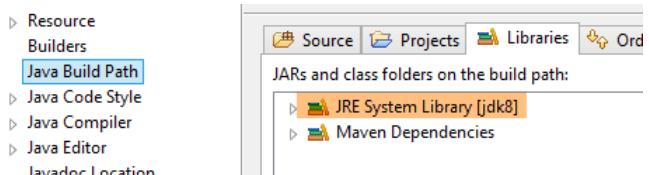
- en [1] et [2], nous importons des projets Maven ;
- en [3a], nous désignons le dossier parent des deux projets à importer ;



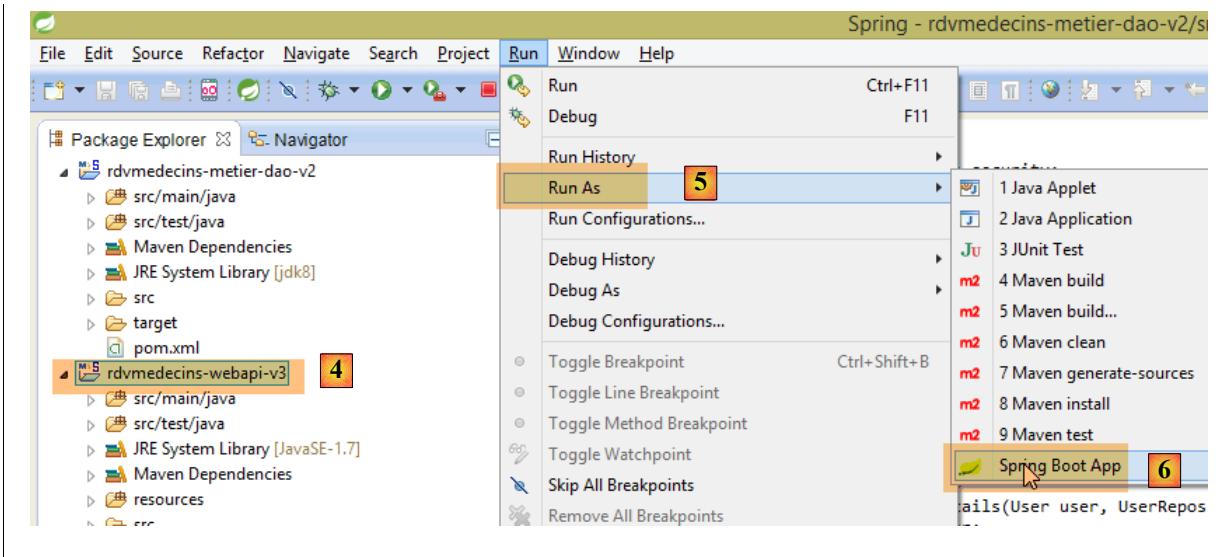
- en [3b], les projets importés. Il se peut que les projets présentent des erreurs. Il faut que chacun d'eux utilise un compilateur ≥ 1.7 :



Il faut donc une JVM ≥ 1.7 :



Lorsqu'il n'y a plus d'erreurs de JVM, on peut exécuter le projet [rdvmedecins-webapi-v3] :



- en [4], [5] et [6] nous exécutons le projet [rdvmedecins-webapi-v3] comme une application [Spring Boot](#) (il faut que le SGBD MySQL soit lancé) ;

Nous obtenons alors les logs suivants dans la console de STS :

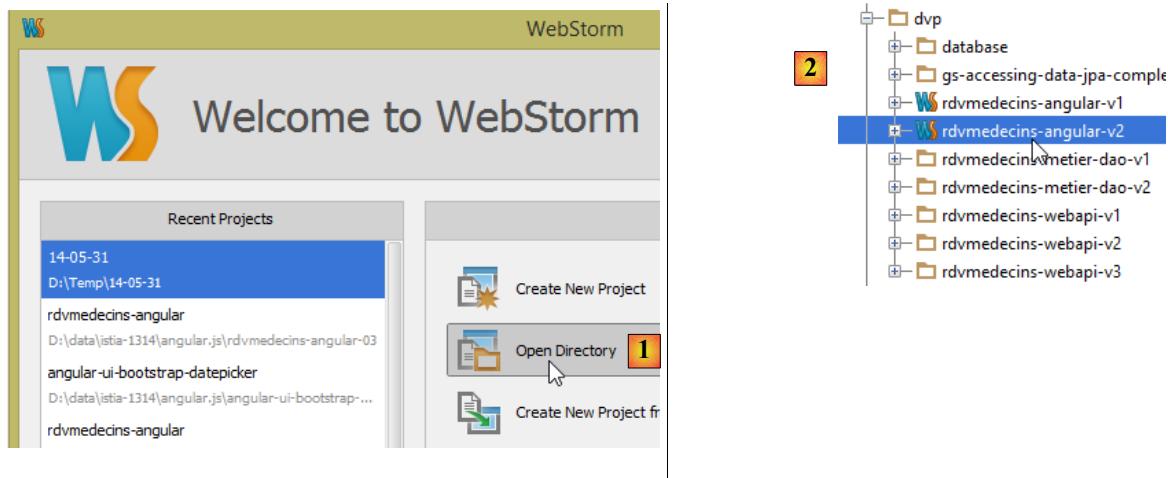
```

1. .
2. \\\ / _` - - - ( _ ) - - - \ \ \ \
3. ( ( ) \ _ [ - - - | - - - | , - - - \ \ \ \
4. \ \ \ ( _ ) | ( ) | [ - - - | ( _ | ) ) ) )
5. ' [ _ ] . _ | [ _ ] [ _ ] , [ / / /
6. =====|_|=====|_|=/_/_/
7. :: Spring Boot ::      (v1.0.0.RELEASE)
8.
9. 2014-06-05 12:22:34.049  INFO 9296 --- [           main] rdvmedecins.web.boot.Boot          :
Starting Boot on Gportpers3 with PID 9296 (D:\data\istia-1314\polys\istia\angularjs-spring4\rdvmedecins-
webapi\target\classes started by ST)
10. 2014-06-05 12:22:34.122  INFO 9296 --- [           main] ationConfigEmbeddedWebApplicationContext :
Refreshing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@4b4bee22:
startup date [Thu Jun 05 12:22:34 CEST 2014]; root of context hierarchy
11. 2014-06-05 12:22:35.083  INFO 9296 --- [           main] o.s.b.f.s.DefaultListableBeanFactory    :
Overriding bean definition for bean 'org.springframework.boot.autoconfigure.AutoConfigurationPackages':
replacing [Generic bean: class
[org.springframework.boot.autoconfigure.AutoConfigurationPackages$BasePackages]; scope=; abstract=false;
lazyInit=false; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false;
factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null] with [Generic
bean: class [org.springframework.boot.autoconfigure.AutoConfigurationPackages$BasePackages]; scope=;
abstract=false; lazyInit=false; autowireMode=0; dependencyCheck=0; autowireCandidate=true;
primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null;
destroyMethodName=null]
12. ...
13. s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080/http
14. 2014-06-05 12:22:41.630  INFO 9296 --- [           main] rdvmedecins.web.boot.Boot : Started Boot in 8.0
seconds (JVM running for 8.944)
```

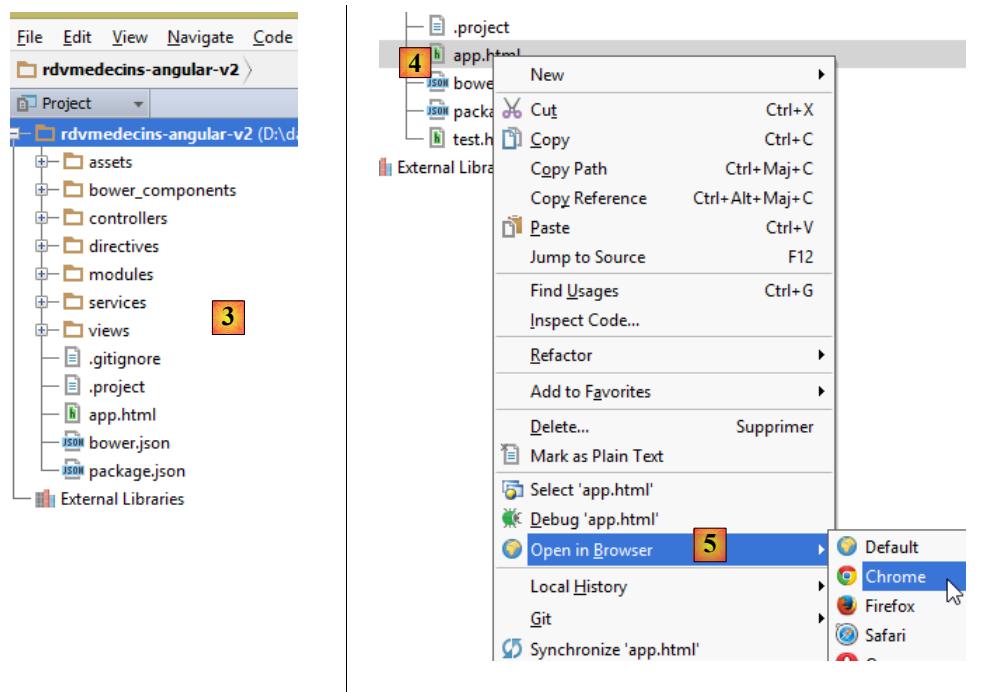
- lignes 13-14 : l'application a démarré sur un serveur Tomcat.

1.3.3 Mise en oeuvre du client Angular

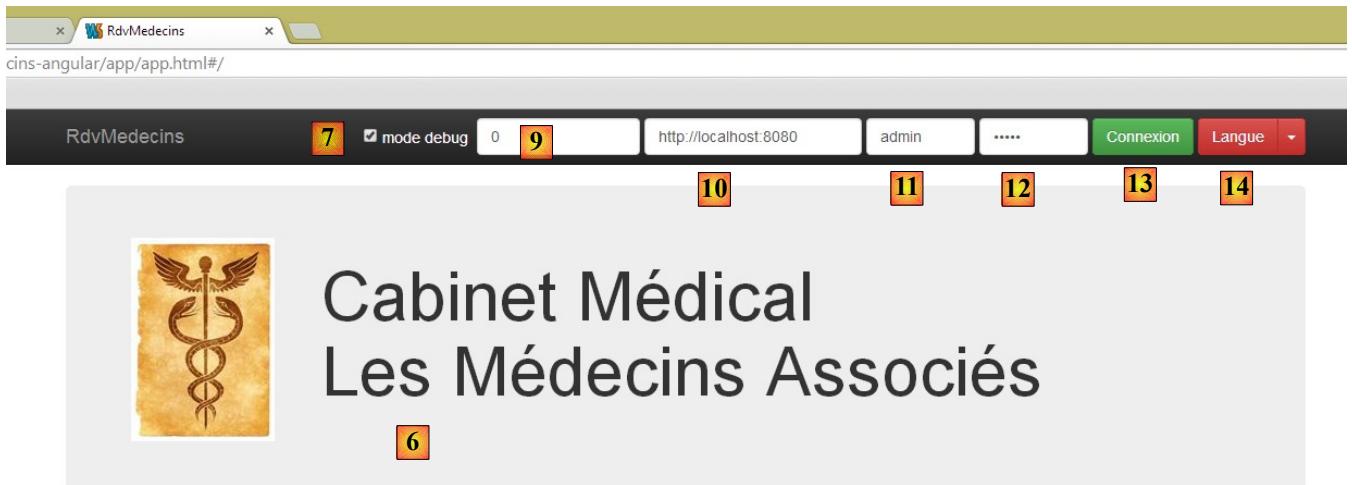
Nous ouvrons le dossier [rdvmedecins-angular-v2] avec WebStorm :



- en [1], on choisit l'option [Open Directory] ;
- en [2], on sélectionne le dossier [rdvmedecins-angular-v2] ;



- en [3], l'arborescence du dossier ;
- en [4], on sélectionne la page principale [app.html] de l'application ;
- en [5], on l'ouvre dans un navigateur récent ;



Authentifiez-vous pour accéder à l'application.

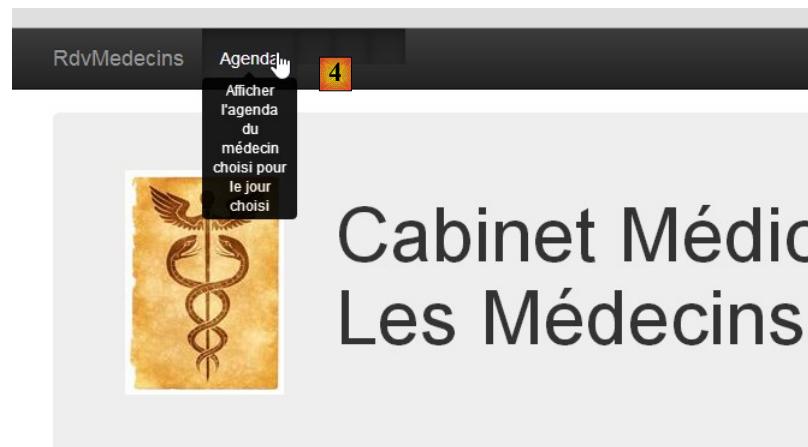
Pour connaître le rôle des différentes saisies, passez le curseur dessus pour avoir une aide.

```
Modèle : { "waitingTimeBeforeTask": 0, "delay": { "on": true }, "titre": { "text": "identification", "show": true, "model": {} }, "navbarrun": { "show": false }, "navbarstart": { "show": true }, "errors": { "show": false }, "view": { "url": "/identification", "model": {}, "done": false }, "waiting": { "text": "msg_waiting_init", "show": false }, "task": { "action": { "promise": {} }, "isFinished": false }, "serverUrl": "http://localhost:8080", "username": "admin", "password": "admin" }
```

- en [6], la page d'entrée de l'application. Il s'agit d'une application de prise de rendez-vous pour des médecins. Cette application a déjà été traitée dans le document [Introduction aux frameworks JSF2, Primefaces et Primefaces mobile](#) ;
- en [7], une case à cocher qui permet d'être ou non en mode [debug]. Ce dernier se caractérise par la présence du cadre [8] qui affiche le modèle de la vue courante ;
- en [9], une durée d'attente artificielle en millisecondes. Elle vaut 0 par défaut (pas d'attente). Si N est la valeur de ce temps d'attente, toute action de l'utilisateur sera exécutée après un temps d'attente de N millisecondes. Cela permet de voir la gestion de l'attente mise en place par l'application ;
- en [10], l'URL du serveur Spring 4. Si on suit ce qui a précédé, c'est [http://localhost:8080] ;
- en [11] et [12], l'identifiant et le mot de passe de celui qui veut utiliser l'application. Il y a deux utilisateurs : **admin/admin** (login/password) avec un rôle (ADMIN) et **user/user** avec un rôle (USER). Seul le rôle ADMIN a le droit d'utiliser l'application. Le rôle USER n'est là que pour montrer ce que répond le serveur dans ce cas d'utilisation ;
- en [13], le bouton qui permet de se connecter au serveur ;
- en [14], la langue de l'application. Il y en a deux : le français par défaut et l'anglais.



- en [1], on se connecte ;



This screenshot shows the selection process. On the left, under "Médecin" [2], a dropdown menu is open, showing "Mme Marie PELISSIER" as the selected option. On the right, under "Jour" [3], a calendar for June 2014 is displayed, with the date "05" highlighted in a blue box. The days of the week are labeled in French: dim., lun., mar., mer., jeu., ven., sam.

	dim.	lun.	mar.	mer.	jeu.	ven.	sam.
22	01	02	03	04	05	06	07
23	08	09	10	11	12	13	14
24	15	16	17	18	19	20	21
25	22	23	24	25	26	27	28
26	29	30	01	02	03	04	05
27	06	07	08	09	10	11	12

- une fois connecté, on peut choisir le médecin avec lequel on veut un rendez-vous [2] et le jour de celui-ci [3] ;
- on demande en [4] à voir l'agenda du médecin choisi pour le jour choisi ;



Cabinet Médical Les Médecins

Rendez - vous de Mme Marie PELISSIER le 2014-06-05

Créneau horaire	Client	Action
08h00:08h20		Réserver  
08h20:08h40		Réserver 
08h40:09h00		Réserver
09h00:09h20		Réserver

- une fois obtenu l'agenda du médecin, on peut réserver un créneau [5] ;



Cabinet Médical Les Médecins Ass

Rendez - vous de Mme Marie PELISSIER le 2014-06-05 à 08h00:08h20

Client 

Mr Jules MARTIN 

- en [6], on choisit le patient pour le rendez-vous et on valide ce choix en [7] ;



Cabinet Médical Les Médecins Associés

Rendez - vous de Mme Marie PELISSIER le 2014-06-05

Créneau horaire	Client	Action
08h00:08h20	Mr Jules MARTIN	Supprimer Supprimer le rendez-vous [7]
08h20:08h40		Réserver
08h40:09h00		Réserver
09h00:09h20		Réserver

Une fois le rendez-vous validé, on est ramené automatiquement à l'agenda où le nouveau rendez-vous est désormais inscrit. Ce rendez-vous pourra être ultérieurement supprimé [7].

Les principales fonctionnalités ont été décrites. Elles sont simples. Celles qui n'ont pas été décrites sont des fonctions de navigation pour revenir à une vue précédente. Terminons par la gestion de la langue :

RdvMedecins Agenda mode debug 0 Déconnexion Langue

French English [1]

Cabinet Médical Les Médecins Associés

Choisissez un médecin et un jour pour avoir l'agenda

Médecin

Mme Marie PELISSIER

Jour

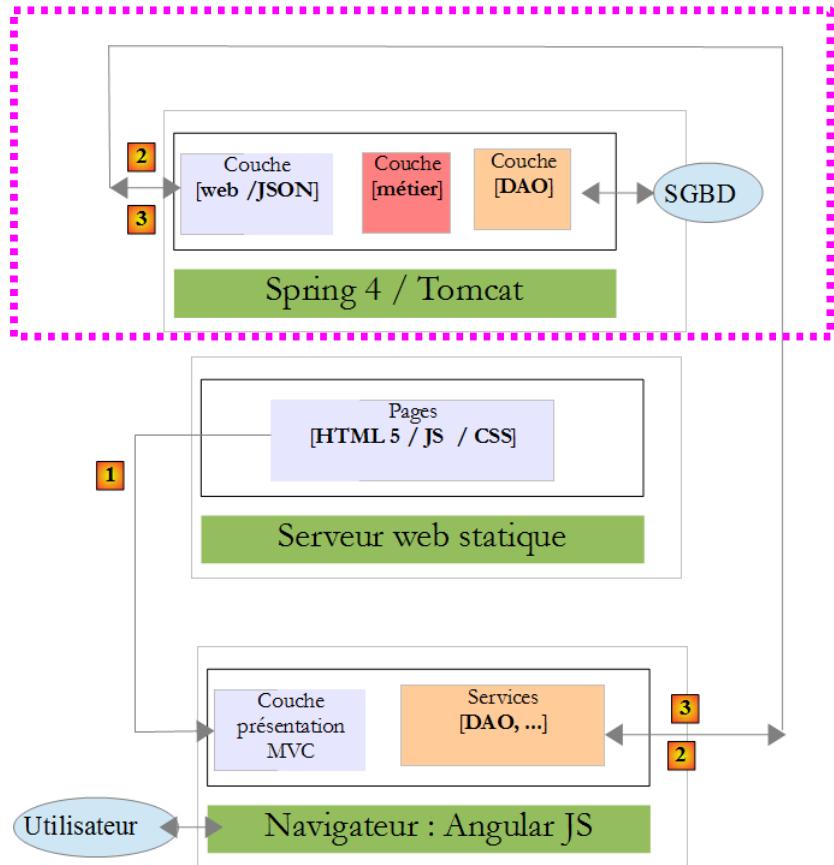
juin 2014						
dim.	lun.	mar.	mer.	jeu.	ven.	sam.
22	01	02	03	04	05	06
23	08	09	10	11	12	13
24	15	16	17	18	19	20
25	22	23	24	25	26	27
26	29	30	01	02	03	04
27	06	07	08	09	10	11
						12

- en [1], on passe du français à l'anglais ;

The screenshot shows a web application interface for managing appointments. At the top, there is a dark header bar with the following elements from left to right: "RdvMedecins", "Show the diary", a search bar containing "debug mode 0", a "Logout" button, and a "Language" dropdown menu set to "English". Below the header is a large title "The Associated Doctors" in a bold, dark font. To the left of the title is a stylized caduceus logo. In the center, there is a small red square containing the number "2", likely indicating the count of appointments. Below the title is a light blue horizontal bar with the text "Select a doctor and a date". Underneath this bar, there are two main sections: "Doctor" on the left and "Date" on the right. The "Doctor" section contains a dropdown menu currently set to "Mme Marie PELISSIER". The "Date" section displays a calendar for June 2014, showing days from Sunday to Saturday. The 5th of June is highlighted with a blue background, while other dates are shown in grey. The entire interface is presented in English.

- en [2], la vue est passée en anglais, y-compris le calendrier ;

2 Le serveur Spring 4

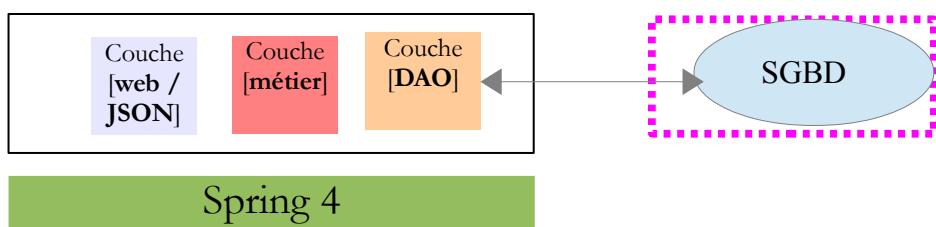


Dans l'architecture ci-dessus, nous abordons maintenant la construction du service web / JSON construit avec le framework Spring 4. Nous allons l'écrire en plusieurs étapes :

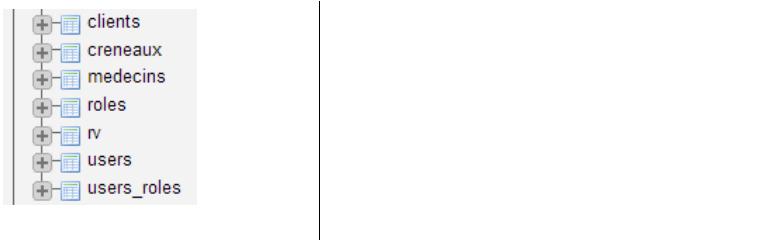
- d'abord les couches [métier] et [DAO] (Data Access Object). Nous utiliserons ici Spring Data ;
- puis le service web JSON sans authentification. Nous utiliserons ici Spring MVC ;
- puis on ajoutera la partie authentification avec Spring Security.

Nous commençons par expliciter la structure de la base de données sous-tendant l'application.

2.1 La base de données



La base de données appelée par la suite [dbrdvmedecins] est une base de données MySQL5 avec les tables suivantes :

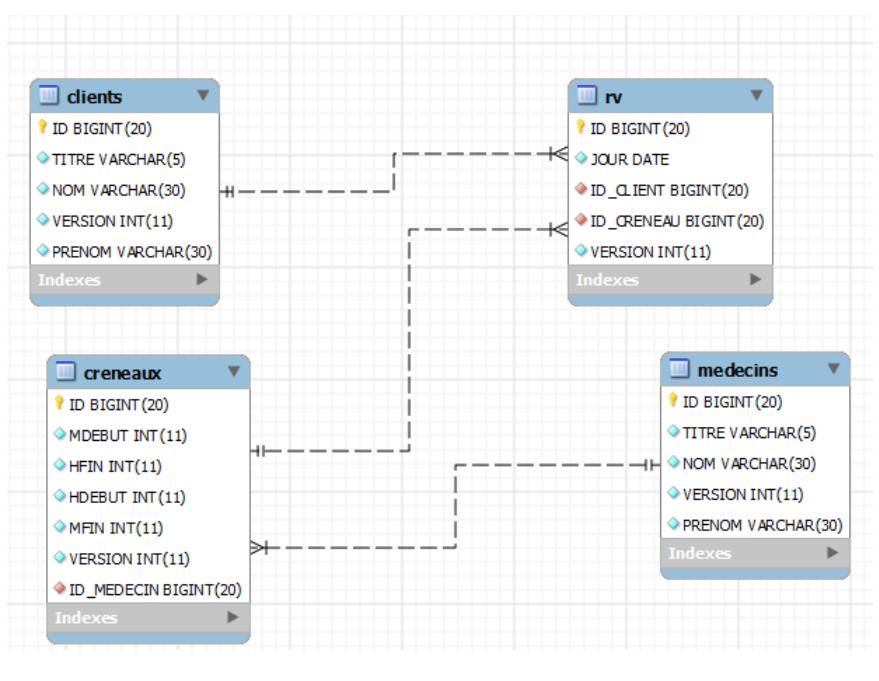


Les rendez-vous sont gérés par les tables suivantes :

- [medecins] : contient la liste des médecins du cabinet ;
- [clients] : contient la liste des patients du cabinet ;
- [creneaux] : contient les créneaux horaires de chacun des médecins ;
- [rv] : contient la liste des rendez-vous des médecins.

Les tables [roles], [users] et [users_roles] sont des tables liées à l'authentification. Dans un premier temps, nous n'allons pas nous en occuper.

Le relations entre les tables gérant les rendez-vous sont les suivantes :



- un créneau horaire appartient à un médecin – un médecin a 0 ou plusieurs créneaux horaires ;
- un rendez-vous réunit à la fois un client et un médecin via un créneau horaire de ce dernier ;
- un client a 0 ou plusieurs rendez-vous ;
- à un créneau horaire est associé 0 ou plusieurs rendez-vous (à des jours différents).

2.1.1 La table [MEDECINS]

Elle contient des informations sur les médecins gérés par l'application [RdvMedecins].

Fields					Indices	Foreign Keys	Data	Description	DDL
Field Name	Field Type	Size	Precision	Not Null					
ID	BIGINT	20	0	✓					
VERSION	INTEGER	11	0	✓					
TITRE	VARCHAR	5	0	✓					
NOM	VARCHAR	30	0	✓					
PRENOM	VARCHAR	30	0	✓					

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mme	PELISSIER	Marie
2	1	Mr	BROMARD	Jacques
3	1	Mr	JANDOT	Philippe
4	1	Melle	JACQUEMOT	Justine

- ID : n° identifiant le médecin - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du médecin
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

2.1.2 La table [CLIENTS]

Les clients des différents médecins sont enregistrés dans la table [CLIENTS] :

Fields Indices Foreign Keys Data Description DDL					
Field Name	Field Type	Size	Precision	Not Null	
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>	
TITRE	VARCHAR	5	0	<input checked="" type="checkbox"/>	
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	
PRENOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mr	MARTIN	Jules
2	1	Mme	GERMAN	Christine
3	1	Mr	JACQUARD	Jules
4	1	Melle	BISTRON	Brigitte

- ID : n° identifiant le client - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du client
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

2.1.3 La table [CRENEAUX]

Elle liste les créneaux horaires où les RV sont possibles :

Fields Indices Foreign Keys Data Description DDL						
Field Name	Field Type	Size	Precision	Not Null	Default	
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	Null	
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>		
HDEBUT	INTEGER	11	0	<input checked="" type="checkbox"/>		
MDEBUT	INTEGER	11	0	<input checked="" type="checkbox"/>		
HFIN	INTEGER	11	0	<input checked="" type="checkbox"/>		
MFIN	INTEGER	11	0	<input checked="" type="checkbox"/>		
ID_MEDECIN	BIGINT	20	0	<input checked="" type="checkbox"/>		

ID	VERSION	ID_MEDECIN	HDEBUT	MDEBUT	HFIN	MFIN
1	1	1	8	0	8	20
2	1	1	8	20	8	40
3	1	1	8	40	9	0
4	1	1	9	0	9	20
5	1	1	9	20	9	40
6	1	1	9	40	10	0
7	1	1	10	0	10	20
8	1	1	10	20	10	40
9	1	1	10	40	11	0
10	1	1	11	0	11	20
11	1	1	11	20	11	40
12	1	1	11	40	12	0
13	1	1	14	0	14	20
14	1	1	14	20	14	40
15	1	1	14	40	15	0
1						
16	1	1	15	0	15	20
17	1	1	15	20	15	40
18	1	1	15	40	16	0
19	1	1	16	0	16	20
20	1	1	16	20	16	40
21	1	1	16	40	17	0
22	1	1	17	0	17	20
23	1	1	17	20	17	40
24	1	1	17	40	18	0
25	1	2	8	0	8	20
26	1	2	8	20	8	40
27	1	2	8	40	9	0
28	1	2	9	0	9	20
29	1	2	9	20	9	40
30	1	2	9	40	10	0
31	1	2	10	0	10	20
32	1	2	10	20	10	40
33	1	2	10	40	12	0
34	1	2	12	0	12	20
35	1	2	12	20	12	40
36	1	2	12	40	12	0
37	1	3	8	0	8	20
38	1	3	8	20	8	40
39	1	3	8	40	9	0
40	1	3	9	0	9	20
41	1	3	9	20	9	40
42	1	3	9	40	10	0
43	1	3	10	0	10	20
44	1	3	10	20	10	40
45	1	3	10	40	12	0
46	1	3	12	0	12	20

- ID : n° identifiant le créneau horaire - clé primaire de la table (ligne 8)
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- ID_MEDECIN : n° identifiant le médecin auquel appartient ce créneau – clé étrangère sur la colonne MEDECINS(ID).
- HDEBUT : heure début créneau
- MDEBUT : minutes début créneau

- HFIN : heure fin créneau
- MFIN : minutes fin créneau

La seconde ligne de la table [CRENEAUX] (cf [1] ci-dessus) indique, par exemple, que le créneau n° 2 commence à 8 h 20 et se termine à 8 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER).

2.1.4 La table [RV]

Elle liste les RV pris pour chaque médecin :

Fields	Indices	Foreign Keys	Data	Description	DDL
Field Name	Field Type	Size	Precision	Not Null	Default
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	Null
JOUR	DATE	10	0	<input checked="" type="checkbox"/>	
ID_CLIENT	BIGINT	20	0	<input checked="" type="checkbox"/>	
ID_CRENEAU	BIGINT	20	0	<input checked="" type="checkbox"/>	

ID	JOUR	ID_CLIENT	ID_CRENEAU
1	22/08/2006	2	1
3	23/08/2006	4	20
4	10/09/2006	2	10
6	23/08/2006	3	7
9	23/08/2006	2	10
1			

- ID : n° identifiant le RV de façon unique – clé primaire
- JOUR : jour du RV
- ID_CRENEAU : créneau horaire du RV - clé étrangère sur le champ [ID] de la table [CRENEAUX] – fixe à la fois le créneau horaire et le médecin concerné.
- ID_CLIENT : n° du client pour qui est faite la réservation – clé étrangère sur le champ [ID] de la table [CLIENTS]

Cette table a une contrainte d'unicité sur les valeurs des colonnes jointes (JOUR, ID_CRENEAU) :

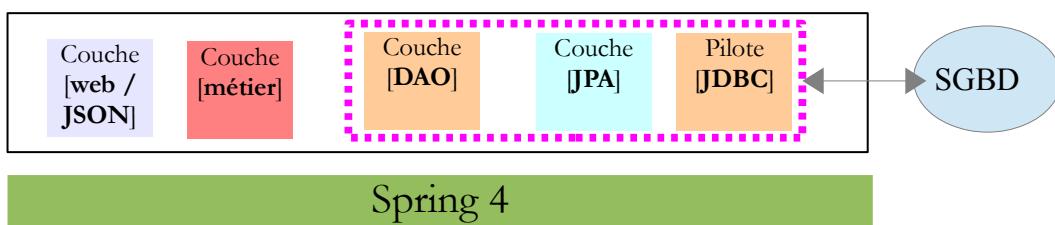
```
ALTER TABLE RV ADD CONSTRAINT UNQ1_RV UNIQUE (JOUR, ID_CRENEAU);
```

Si une ligne de la table [RV] a la valeur (JOUR1, ID_CRENEAU1) pour les colonnes (JOUR, ID_CRENEAU), cette valeur ne peut se retrouver nulle part ailleurs. Sinon, cela signifierait que deux RV ont été pris au même moment pour le même médecin. D'un point de vue programmation Java, le pilote JDBC de la base lance une *SQLException* lorsque ce cas se produit.

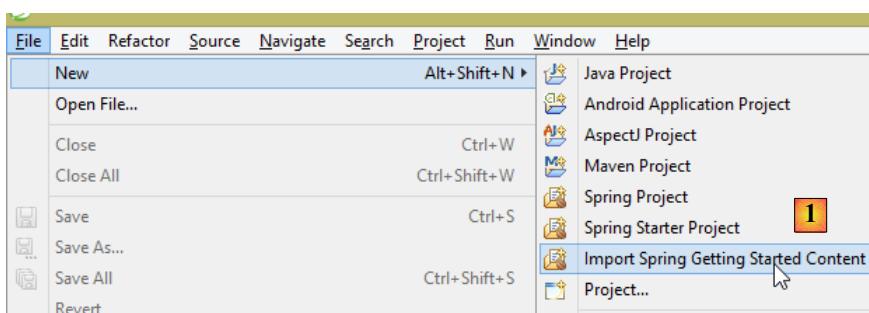
La ligne d'*id* égal à 3 (cf [1] ci-dessus) signifie qu'un RV a été pris pour le créneau n° 20 et le client n° 4 le 23/08/2006. La table [CRENEAUX] nous apprend que le créneau n° 20 correspond au créneau horaire 16 h 20 - 16 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER). La table [CLIENTS] nous apprend que le client n° 4 est Melle Brigitte BISTROU.

2.2 Introduction à Spring Data

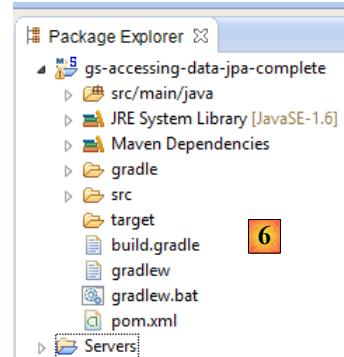
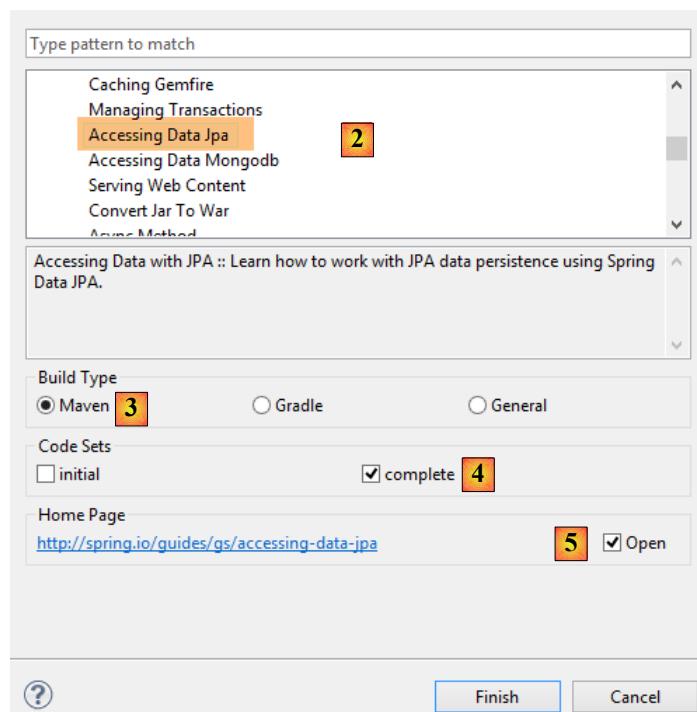
Nous allons implémenter la couche [DAO] du projet avec Spring Data, une branche de l'écosystème Spring.



Sur le site de Spring existent de nombreux tutoriels pour démarrer avec Spring [<http://spring.io/guides>]. Nous allons utiliser l'un d'eux pour introduire Spring Data. Nous utilisons pour cela Spring Tool Suite (STS).



- en [1], nous importons l'un des tutoriels de [spring.io/guides] ;



- en [2], on choisit le tutoriel [Accessing Data Jpa] qui montre comment accéder à une base de données avec Spring Data ;
- en [3], on choisit un projet configuré par Maven ;
- en [4], le tutoriel peut être délivré sous deux formes : [initial] qui est une version vide qu'on remplit en suivant le tutoriel ou [complete] qui est la version finale du tutoriel. Nous choisissons cette dernière ;
- en [5], on peut choisir de visualiser le tutoriel dans un navigateur ;
- en [6], le projet final.

2.2.1 La configuration Maven du projet

Les dépendances Maven du projet sont configurées dans le fichier [pom.xml] :

```

1. <groupId>org.springframework</groupId>
2. <artifactId>gs-accessing-data-jpa</artifactId>
3. <version>0.1.0</version>
4.
5. <parent>
6.   <groupId>org.springframework.boot</groupId>
7.   <artifactId>spring-boot-starter-parent</artifactId>
8.   <version>1.0.2.RELEASE</version>
9. </parent>
10.
11. <dependencies>
```

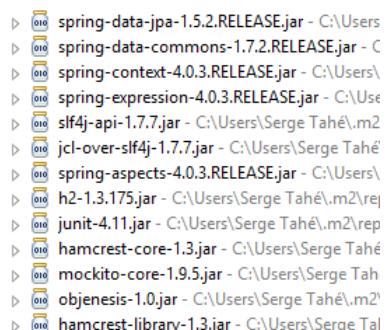
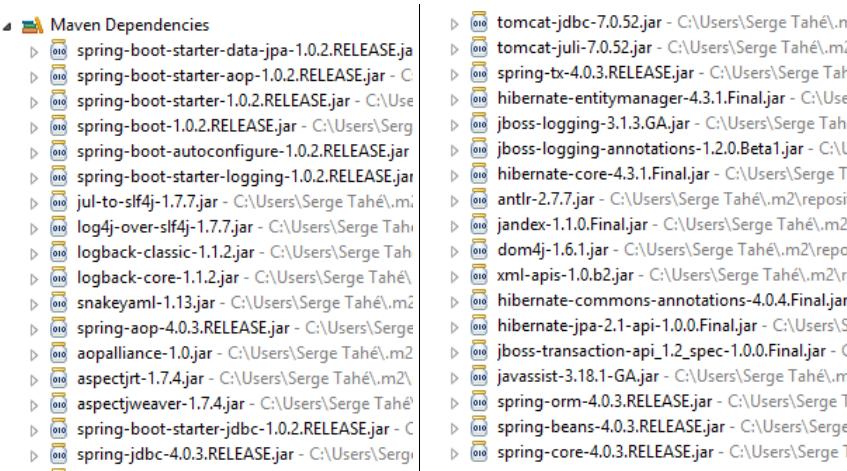
```

12.      <dependency>
13.          <groupId>org.springframework.boot</groupId>
14.          <artifactId>spring-boot-starter-data-jpa</artifactId>
15.      </dependency>
16.      <dependency>
17.          <groupId>com.h2database</groupId>
18.          <artifactId>h2</artifactId>
19.      </dependency>
20.  </dependencies>
21.
22.  <properties>
23.      <!-- use UTF-8 for everything -->
24.      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
25.      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
26.      <start-class>hello.Application</start-class>
27. </properties>

```

- lignes 5-9 : définissent un projet Maven parent. C'est lui qui définit l'essentiel des dépendances du projet. Elles peuvent être suffisantes, auquel cas on n'en rajoute pas, ou pas, auquel cas on rajoute les dépendances manquantes ;
- lignes 12-15 : définissent une dépendance sur [spring-boot-starter-data-jpa]. Cet artifact contient les classes de Spring Data ;
- lignes 16-19 : définissent une dépendance sur le SGBD H2 qui permet de créer et gérer des bases de données en mémoire.

Regardons les classes amenées par ces dépendances :



Elles sont très nombreuses :

- certaines appartiennent à l'écosystème Spring (celles commençant par spring) ;
- d'autres appartiennent à l'écosystème Hibernate (hibernate, jboss) dont on utilise ici l'implémentation JPA ;
- d'autres sont des bibliothèques de tests (junit, hamcrest) ;
- d'autres des bibliothèques de logs (log4j, logback, slf4j) ;

Nous allons les garder toutes. Pour une application en production, il faudrait ne garder que celles qui sont nécessaires.

Ligne 26 du fichier [pom.xml] on trouve la ligne :

```
<start-class>hello.Application</start-class>
```

Cette ligne est liée aux lignes suivantes :

```

1.  <build>
2.      <plugins>
3.          <plugin>
4.              <groupId>org.springframework.boot</groupId>
5.              <artifactId>maven-compiler-plugin</artifactId>
6.          </plugin>
7.          <plugin>
8.              <groupId>org.springframework.boot</groupId>
9.              <artifactId>spring-boot-maven-plugin</artifactId>
9.          </plugin>

```

```

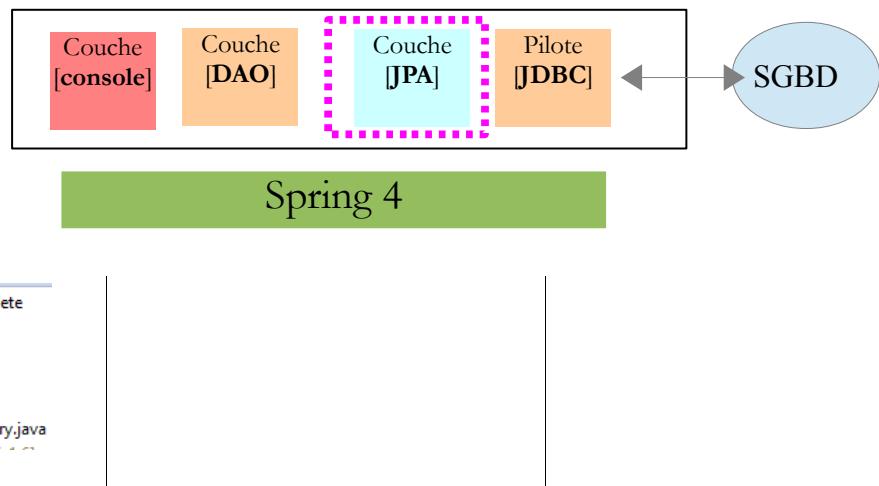
10.      </plugins>
11.    </build>

```

Lignes 6-9, le plugin [spring-boot-maven-plugin] permet de générer le jar exécutable de l'application. La ligne 26 du fichier [pom.xml] désigne alors la classe exécutable de ce jar.

2.2.2 La couche [JPA]

L'accès à la base de données se fait au travers d'une couche [JPA], Java Persistence API :



L'application est basique et gère des clients [Customer]. La classe [Customer] fait partie de la couche [JPA] et est la suivante :

```

1. package hello;
2.
3. import javax.persistence.Entity;
4. import javax.persistence.GeneratedValue;
5. import javax.persistence.GenerationType;
6. import javax.persistence.Id;
7.
8. @Entity
9. public class Customer {
10.
11.     @Id
12.     @GeneratedValue(strategy = GenerationType.AUTO)
13.     private long id;
14.     private String firstName;
15.     private String lastName;
16.
17.     protected Customer() {
18.     }
19.
20.     public Customer(String firstName, String lastName) {
21.         this.firstName = firstName;
22.         this.lastName = lastName;
23.     }
24.
25.     @Override
26.     public String toString() {
27.         return String.format("Customer[id=%d, firstName='%s', lastName='%s']", id, firstName, lastName);
28.     }
29.
30. }

```

Un client a un identifiant [id], un prénom [firstName] et un nom [lastName]. Chaque instance [Customer] représente une ligne d'une table de la base de données.

- ligne 8 : annotation JPA qui fait que la persistance des instances [Customer] (Create, Read, Update, Delete) va être gérée par une implémentation JPA. D'après les dépendances Maven, on voit que c'est l'implémentation JPA / Hibernate qui est utilisée ;

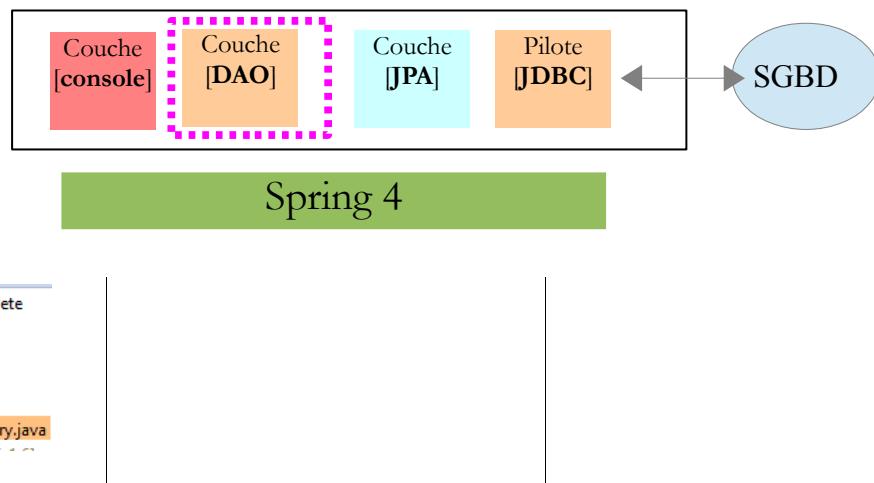
- lignes 11-12 : annotations JPA qui associent le champ [id] à la clé primaire de la table des [Customer]. La ligne 12, indique que l'implémentation JPA utilisera la méthode de génération de clé primaire propre au SGBD utilisé, ici H2 ;

Il n'y a pas d'autres annotations JPA. Des valeurs par défaut seront alors utilisées :

- la table des [Customer] portera le nom de la classe, c-à-d [Customer] ;
- les colonnes de cette table porteront le nom des champs de la classe : [id, firstName, lastName] sachant que la casse n'est pas prise en compte dans le nom d'une colonne de table ;

On notera qu'à aucun moment, l'implémentation JPA utilisée n'est nommée.

2.2.3 La couche [DAO]



La classe [CustomerRepository] implémente la couche [DAO]. Son code est le suivant :

```

1. package hello;
2.
3. import java.util.List;
4.
5. import org.springframework.data.repository.CrudRepository;
6.
7. public interface CustomerRepository extends CrudRepository<Customer, Long> {
8.
9.     List<Customer> findByLastName(String lastName);
10. }
```

C'est donc une interface et non une classe (ligne 7). Elle étend l'interface [CrudRepository], une interface de Spring Data (ligne 5). Cette interface est paramétrée par deux types : le premier est le type des éléments gérés, ici le type [Customer], le second le type de la clé primaire des éléments gérés, ici un type [Long]. L'interface [CrudRepository] est la suivante :

```

1. package org.springframework.data.repository;
2.
3. import java.io.Serializable;
4.
5. @NoRepositoryBean
6. public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {
7.
8.     <S extends T> S save(S entity);
9.
10.    <S extends T> Iterable<S> save(Iterable<S> entities);
11.
12.    T findOne(ID id);
13.
14.    boolean exists(ID id);
15.
16.    Iterable<T> findAll();
17.
18.    Iterable<T> findAll(Iterable<ID> ids);
19.
20.    long count();
```

```

21.
22.     void delete(ID id);
23.
24.     void delete(T entity);
25.
26.     void delete(Iterable<? extends T> entities);
27.
28.     void deleteAll();
29. }
```

Cette interface définit les opération CRUD (Create – Read – Update – Delete) qu'on peut faire sur un type JPA T :

- ligne 8 : la méthode **save** permet de **persister** une entité T en base. Elle rend l'entité persistée avec la clé primaire que lui a donnée le SGBD. Elle permet également de **mettre à jour** une entité T identifiée par sa clé primaire *id*. Le choix de l'une ou l'autre action se fait selon la valeur de la clé primaire *id* : si celle-ci vaut **null** c'est l'opération de persistance qui a lieu, sinon c'est l'opération de mise à jour ;
- ligne 10 : idem mais pour une liste d'entités ;
- ligne 12 : la méthode **findOne** permet de retrouver une entité T identifiée par sa clé primaire *id* ;
- ligne 22 : la méthode **delete** permet de supprimer une entité T identifiée par sa clé primaire *id* ;
- lignes 24-28 : des variantes de la méthode [delete] ;
- ligne 16 : la méthode [findAll] permet de retrouver toutes les entités persistées T ;
- ligne 18 : idem mais limitée aux entités dont on a passé la liste des identifiants ;

Revenons à l'interface [CustomerRepository] :

```

1. package hello;
2.
3. import java.util.List;
4.
5. import org.springframework.data.repository.CrudRepository;
6.
7. public interface CustomerRepository extends CrudRepository<Customer, Long> {
8.
9.     List<Customer> findByLastName(String lastName);
10. }
```

- la ligne 9 permet de retrouver un [Customer] par son nom [lastName] ;

Et c'est tout pour la couche [DAO]. Il n'y a pas de classe d'implémentation de l'interface précédente. Celle-ci est générée à l'exécution par [Spring Data]. Les méthodes de l'interface [CrudRepository] sont automatiquement implémentées. Pour les méthodes rajoutées dans l'interface [CustomerRepository], ça dépend. Revenons à la définition de [Customer] :

```

1.     private long id;
2.     private String firstName;
3.     private String lastName;
```

La méthode de la ligne 9 est implémentée automatiquement par [Spring Data] parce qu'elle référence le champ [lastName] (ligne 3) de [Customer]. Lorsqu'il rencontre une méthode [findBySomething] dans l'interface à implémenter, Spring Data l'implémente par la requête JPQL (Java Persistence Query Language) suivante :

```
select t from T t where t.something=:value
```

Il faut donc que le type T ait un champ nommé [something]. Ainsi la méthode

```
List<Customer> findByLastName(String lastName);
```

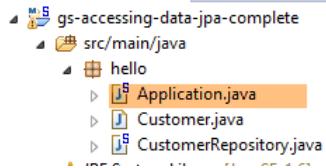
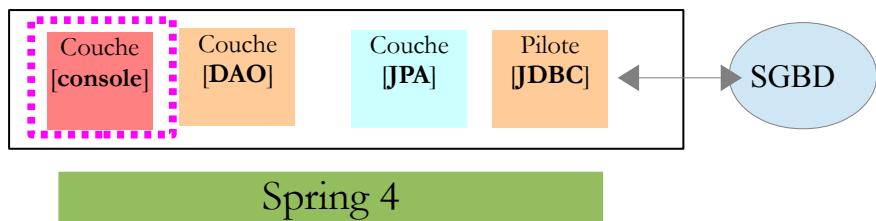
va être implémentée par un code ressemblant au suivant :

```
return [em].createQuery("select c from Customer c where
c.lastName=:value").setParameter("value",lastName).getResultList()
```

où [em] désigne le contexte de persistance JPA. Cela n'est possible que si la classe [Customer] a un champ nommé [lastName], ce qui est le cas.

En conclusion, dans les cas simples, Spring Data nous permet d'implémenter la couche [DAO] avec une simple interface.

2.2.4 La couche [console]



La classe [Application] est la suivante :

```
1. package hello;
2.
3. import java.util.List;
4.
5. import org.springframework.boot.SpringApplication;
6. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
7. import org.springframework.context.ConfigurableApplicationContext;
8. import org.springframework.context.annotation.Configuration;
9.
10. @Configuration
11. @EnableAutoConfiguration
12. public class Application {
13.
14.     public static void main(String[] args) {
15.
16.         ConfigurableApplicationContext context = SpringApplication.run(Application.class);
17.         CustomerRepository repository = context.getBean(CustomerRepository.class);
18.
19.         // save a couple of customers
20.         repository.save(new Customer("Jack", "Bauer"));
21.         repository.save(new Customer("Chloe", "O'Brian"));
22.         repository.save(new Customer("Kim", "Bauer"));
23.         repository.save(new Customer("David", "Palmer"));
24.         repository.save(new Customer("Michelle", "Dessler"));
25.
26.         // fetch all customers
27.         Iterable<Customer> customers = repository.findAll();
28.         System.out.println("Customers found with findAll():");
29.         System.out.println("-----");
30.         for (Customer customer : customers) {
31.             System.out.println(customer);
32.         }
33.         System.out.println();
34.
35.         // fetch an individual customer by ID
36.         Customer customer = repository.findOne(1L);
37.         System.out.println("Customer found with findOne(1L):");
38.         System.out.println("-----");
39.         System.out.println(customer);
40.         System.out.println();
41.
42.         // fetch customers by last name
43.         List<Customer> bauers = repository.findByLastName("Bauer");
44.         System.out.println("Customer found with findByLastName('Bauer'):");
45.         System.out.println("-----");
46.         for (Customer bauer : bauers) {
```

```
47.         System.out.println(bauer);
48.     }
49.
50.     context.close();
51. }
52.
53. }
```

- la ligne 10 : indique que la classe sert à configurer Spring. Les versions récentes de Spring peuvent en effet être configurées en Java plutôt qu'en XML. Les deux méthodes peuvent être utilisées simultanément. Dans le code d'une classe ayant l'annotation [Configuration] on trouve normalement des beans Spring, c-à-d des définitions de classe à instancier. Lorsqu'on travaille avec un SGBD, divers beans Spring doivent être définis :
 - un [EntityManagerFactory] qui définit l'implémentation JPA à utiliser ;
 - un [DataSource] qui définit la source de données à utiliser ;
 - un [TransactionManager] qui définit le gestionnaire de transactions à utiliser ;

Ici aucun de ces beans n'est défini.

- la ligne 11 : l'annotation `[EnableAutoConfiguration]` est une annotation provenant du projet [\[Spring Boot\]](#) (lignes 5-6). Cette annotation demande à Spring Boot via la classe `[SpringApplication]` (ligne 16) de configurer l'application en fonction des bibliothèques trouvées dans son Classpath. Parce que les bibliothèques Hibernate sont dans le Classpath, le bean `[entityManagerFactory]` sera implémenté avec Hibernate. Parce que la bibliothèque du SGBD H2 est dans le Classpath, le bean `[dataSource]` sera implémenté avec H2. Dans le bean `[dataSource]`, on doit définir également l'utilisateur et son mot de passe. Ici Spring Boot utilisera l'administrateur par défaut de H2, **sa** sans mot de passe. Parce que la bibliothèque `[spring-tx]` est dans le Classpath, c'est le gestionnaire de transactions de Spring qui sera utilisé.

Par ailleurs, le dossier dans lequel se trouve la classe [Application] va être scanné à la recherche de beans implicitement reconnus par Spring ou définis explicitement par des annotations Spring. Ainsi les classes [Customer] et [CustomerRepository] vont-elles être inspectées. Parce que la première a l'annotation [@Entity] elle sera cataloguée comme entité à gérer par Hibernate. Parce que la seconde étend l'interface [CrudRepository] elle sera enregistrée comme bean Spring.

Examinons les lignes 16-17 du code :

```
1. ConfigurableApplicationContext context = SpringApplication.run(Application.class);
2. CustomerRepository repository = context.getBean(CustomerRepository.class);
```

- ligne 1 : la méthode statique [run] de la classe [SpringApplication] du projet Spring Boot est exécutée. Son paramètre est la classe qui a une annotation [Configuration] ou [EnableAutoConfiguration]. Tout ce qui a été expliqué précédemment va alors se dérouler. Le résultat est un contexte d'application Spring, c-à-d un ensemble de beans gérés par Spring ;
 - ligne 17 : on demande à ce contexte Spring, un bean implémentant l'interface [CustomerRepository]. Nous récupérons ici, la classe générée par Spring Data pour implémenter cette interface.

Les opérations qui suivent ne font qu'utiliser les méthodes du bean implémentant l'interface [CustomerRepository]. On notera ligne 50, que le contexte est fermé. Les résultats console sont les suivants :

```

16. 2014-06-05 16:23:15.692 INFO 11664 --- [           main] org.hibernate.cfg.Environment      :
HHH000206: hibernate.properties not found
17. 2014-06-05 16:23:15.694 INFO 11664 --- [           main] org.hibernate.cfg.Environment      :
HHH000021: Bytecode provider name : javassist
18. 2014-06-05 16:23:15.988 INFO 11664 --- [           main] o.hibernate.annotations.common.Version   :
HCANN000001: Hibernate Commons Annotations {4.0.4.Final}
19. 2014-06-05 16:23:16.078 INFO 11664 --- [           main] org.hibernate.dialect.Dialect       :
HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
20. 2014-06-05 16:23:16.300 INFO 11664 --- [           main] o.h.h.i.ast.ASTQueryTranslatorFactory  :
HHH000397: Using ASTQueryTranslatorFactory
21. 2014-06-05 16:23:16.613 INFO 11664 --- [           main] org.hibernate.tool.hbm2ddl.SchemaExport :
HHH000227: Running hbm2ddl schema export
22. Hibernate: drop table customer if exists
23. Hibernate: create table customer (id bigint generated by default as identity, first_name varchar(255), last_name varchar(255), primary key (id))
24. 2014-06-05 16:23:16.619 INFO 11664 --- [           main] org.hibernate.tool.hbm2ddl.SchemaExport :
HHH000230: Schema export complete
25. 2014-06-05 16:23:17.074 INFO 11664 --- [           main] o.s.j.e.a.AnnotationMBeanExporter      :
Registering beans for JMX exposure on startup
26. 2014-06-05 16:23:17.094 INFO 11664 --- [           main] hello.Application                  :
Started Application in 3.906 seconds (JVM running for 5.013)
27. Hibernate: insert into customer (id, first_name, last_name) values (null, ?, ?)
28. Hibernate: insert into customer (id, first_name, last_name) values (null, ?, ?)
29. Hibernate: insert into customer (id, first_name, last_name) values (null, ?, ?)
30. Hibernate: insert into customer (id, first_name, last_name) values (null, ?, ?)
31. Hibernate: insert into customer (id, first_name, last_name) values (null, ?, ?)
32. Hibernate: select customer0_.id as id1_0_, customer0_.first_name as first_na2_0_, customer0_.last_name as last_nam3_0_ from customer customer0_
33. Customers found with findAll():
34. -----
35. Customer[id=1, firstName='Jack', lastName='Bauer']
36. Customer[id=2, firstName='Chloe', lastName='O'Brian']
37. Customer[id=3, firstName='Kim', lastName='Bauer']
38. Customer[id=4, firstName='David', lastName='Palmer']
39. Customer[id=5, firstName='Michelle', lastName='Dessler']
40.
41. Hibernate: select customer0_.id as id1_0_0_, customer0_.first_name as first_na2_0_0_, customer0_.last_name as last_nam3_0_0_ from customer customer0_ where customer0_.id=?
42. Customer found with findOne(1L):
43. -----
44. Customer[id=1, firstName='Jack', lastName='Bauer']
45.
46. Hibernate: select customer0_.id as id1_0_, customer0_.first_name as first_na2_0_, customer0_.last_name as last_nam3_0_ from customer customer0_ where customer0_.last_name=?
47. Customer found with findByLastName('Bauer'):
48. -----
49. Customer[id=1, firstName='Jack', lastName='Bauer']
50. Customer[id=3, firstName='Kim', lastName='Bauer']
51. 2014-06-05 16:23:17.330 INFO 11664 --- [           main] s.c.a.AnnotationConfigApplicationContext :
Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@331a8fa0: startup date
[Thu Jun 05 16:23:13 CEST 2014]; root of context hierarchy
52. 2014-06-05 16:23:17.332 INFO 11664 --- [           main] o.s.j.e.a.AnnotationMBeanExporter      :
Unregistering JMX-exposed beans on shutdown
53. 2014-06-05 16:23:17.333 INFO 11664 --- [           main] j.LocalContainerEntityManagerFactoryBean :
Closing JPA EntityManagerFactory for persistence unit 'default'
54. 2014-06-05 16:23:17.334 INFO 11664 --- [           main] org.hibernate.tool.hbm2ddl.SchemaExport :
HHH000227: Running hbm2ddl schema export
55. Hibernate: drop table customer if exists
56. 2014-06-05 16:23:17.336 INFO 11664 --- [           main] org.hibernate.tool.hbm2ddl.SchemaExport :
HHH000230: Schema export complete

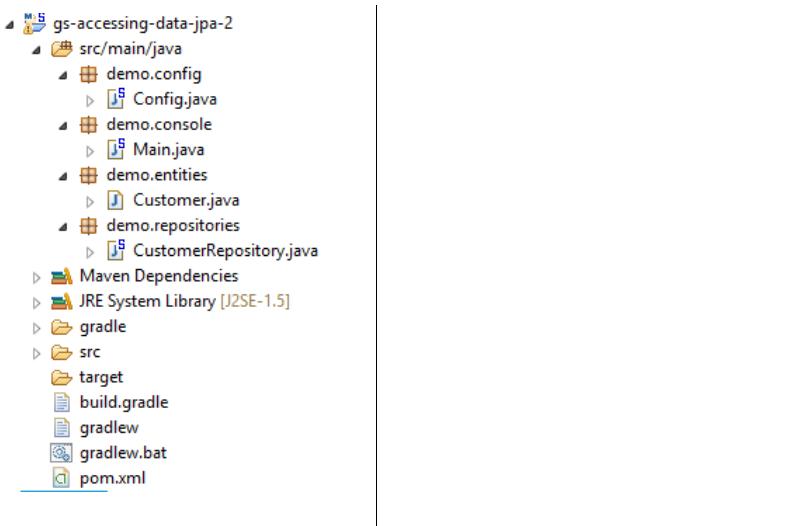
```

- lignes 1-8 : le logo du projet Spring Boot ;
- ligne 9 : la classe [hello.Application] est exécutée ;
- ligne 10 : [AnnotationConfigApplicationContext] est une classe implémentant l'interface [ApplicationContext] de Spring. C'est un conteneur de beans ;
- ligne 11 : le bean [entityManagerFactory] est implémentée avec la classe [LocalContainerEntityManagerFactory], une classe de Spring ;
- ligne 12 : on voit apparaître [hibernate]. C'est cette implémentation JPA qui a été choisie ;
- ligne 19 : un dialecte Hibernate est la variante SQL à utiliser avec le SGBD. Ici le dialecte [H2Dialect] montre qu'Hibernate va travailler avec le SGBD H2 ;
- lignes 22-24 : la table [CUSTOMER] est créée. Cela signifie qu'Hibernate a été configuré pour générer les tables à partir des définitions JPA, ici la définition JPA de la classe [Customer] ;

- lignes 27-32 : des logs d'Hibernate montrant les insertions de lignes dans la table [CUSTOMER]. Cela signifie qu'Hibernate a été configuré pour générer des logs ;
- lignes 35-39 : les cinq clients insérés ;
- lignes 42-44 : résultat de la méthode [findOne] de l'interface ;
- lignes 47-50 : résultats de la méthode [findByName] ;
- lignes 51 et suivantes : logs de la fermeture du contexte Spring.

2.2.5 Configuration manuelle du projet Spring Data

Nous dupliquons le projet précédent dans le projet [gs-accessing-data-jpa-2] :



Dans ce nouveau projet, nous n'allons pas nous reposer sur la configuration automatique faite par Spring Boot. Nous allons la faire manuellement. Cela peut être utile si les configurations par défaut ne nous conviennent pas.

Tout d'abord, nous allons expliciter les dépendances nécessaires dans le fichier [pom.xml] :

```

1. <dependencies>
2.     <!-- Spring Core -->
3.     <dependency>
4.         <groupId>org.springframework</groupId>
5.         <artifactId>spring-core</artifactId>
6.         <version>4.0.5.RELEASE</version>
7.     </dependency>
8.     <dependency>
9.         <groupId>org.springframework</groupId>
10.        <artifactId>spring-context</artifactId>
11.        <version>4.0.5.RELEASE</version>
12.    </dependency>
13.    <dependency>
14.        <groupId>org.springframework</groupId>
15.        <artifactId>spring-beans</artifactId>
16.        <version>4.0.5.RELEASE</version>
17.    </dependency>
18.    <!-- Spring transactions -->
19.    <dependency>
20.        <groupId>org.springframework</groupId>
21.        <artifactId>spring-aop</artifactId>
22.        <version>4.0.5.RELEASE</version>
23.    </dependency>
24.    <dependency>
25.        <groupId>org.springframework</groupId>
26.        <artifactId>spring-tx</artifactId>
27.        <version>4.0.5.RELEASE</version>
28.    </dependency>
29.    <!-- Spring Data -->
30.    <dependency>
31.        <groupId>org.springframework.data</groupId>
32.        <artifactId>spring-data-jpa</artifactId>
33.        <version>1.5.2.RELEASE</version>

```

```

34.      </dependency>
35.      <!-- Spring Boot -->
36.      <dependency>
37.          <groupId>org.springframework.boot</groupId>
38.          <artifactId>spring-boot</artifactId>
39.          <version>1.0.2.RELEASE</version>
40.      </dependency>
41.      <!-- Hibernate -->
42.      <dependency>
43.          <groupId>org.hibernate</groupId>
44.          <artifactId>hibernate-entitymanager</artifactId>
45.          <version>4.3.4.Final</version>
46.      </dependency>
47.      <!-- H2 Database -->
48.      <dependency>
49.          <groupId>com.h2database</groupId>
50.          <artifactId>h2</artifactId>
51.          <version>1.4.178</version>
52.      </dependency>
53.      <!-- Commons DBCP -->
54.      <dependency>
55.          <groupId>commons-dbcp</groupId>
56.          <artifactId>commons-dbcp</artifactId>
57.          <version>1.4</version>
58.      </dependency>
59.      <dependency>
60.          <groupId>commons-pool</groupId>
61.          <artifactId>commons-pool</artifactId>
62.          <version>1.6</version>
63.      </dependency>
64.  </dependencies>

```

- lignes 3-17 : les bibliothèques de base de Spring ;
- lignes 19-28 : les bibliothèques de Spring pour gérer les transactions avec une base de données ;
- lignes 30-34 : Spring Data utilisé pour accéder à la base de données ;
- lignes 36-40 : Spring Boot pour lancer l'application ;
- lignes 48-52 : le SGBD H2 ;
- lignes 54-63 : les bases de données sont souvent utilisées avec des pools de connexions ouvertes qui évitent les ouvertures / fermetures de connexion à répétition. Ici, l'implémentation utilisée est celle de [commons-dbcp] ;

Toujours dans [pom.xml], on modifie le nom de la classe exécutable :

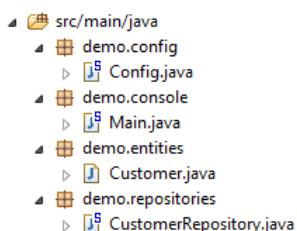
```

1.      <properties>
2.      ...
3.          <start-class>demo.console.Main</start-class>
4.      </properties>

```

Dans le nouveau projet, l'entité [Customer] et l'interface [CustomerRepository] ne changent pas. On va changer la classe [Application] qui va être scindée en deux classes :

- [Config] qui sera la classe de configuration ;
- [Main] qui sera la classe exécutable ;



La classe exécutable [Main] est la même que précédemment sans les annotations de configuration :

```

1. package demo.console;
2.
3. import java.util.List;
4.

```

```

5. import org.springframework.boot.SpringApplication;
6. import org.springframework.context.ConfigurableApplicationContext;
7.
8. import demo.config.Config;
9. import demo.entities.Customer;
10. import demo.repositories.CustomerRepository;
11.
12. public class Main {
13.
14.     public static void main(String[] args) {
15.
16.         ConfigurableApplicationContext context = SpringApplication.run(Config.class);
17.         CustomerRepository repository = context.getBean(CustomerRepository.class);
18.     ...
19.
20.         context.close();
21.     }
22.
23. }

```

- ligne 12 : la classe [Main] n'a plus d'annotations de configuration ;
- ligne 16 : l'application est lancée avec Spring Boot. Le paramètre [Config.class] est la nouvelle classe de configuration du projet ;

La classe [Config] qui configure le projet est la suivante :

```

1. package demo.config;
2.
3. import javax.persistence.EntityManagerFactory;
4. import javax.sql.DataSource;
5.
6. import org.apache.commons.dbcp.BasicDataSource;
7. import org.springframework.context.annotation.Bean;
8. import org.springframework.context.annotation.Configuration;
9. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
10. import org.springframework.orm.jpa.JpaTransactionManager;
11. import org.springframework.orm.jpa.JpaVendorAdapter;
12. import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
13. import org.springframework.orm.jpa.vendor.Database;
14. import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
15. import org.springframework.transaction.PlatformTransactionManager;
16. import org.springframework.transaction.annotation.EnableTransactionManagement;
17.
18. // @ComponentScan(basePackages = { "demo" })
19. // @EntityScan(basePackages = { "demo.entities" })
20. @EnableTransactionManagement
21. @EnableJpaRepositories(basePackages = { "demo.repositories" })
22. @Configuration
23. public class Config {
24.     // la source de données H2
25.     @Bean
26.     public DataSource dataSource() {
27.         BasicDataSource dataSource = new BasicDataSource();
28.         dataSource.setDriverClassName("org.h2.Driver");
29.         dataSource.setUrl("jdbc:h2:./demo");
30.         dataSource.setUsername("sa");
31.         dataSource.setPassword("");
32.         return dataSource;
33.     }
34.
35.     // le provider JPA
36.     @Bean
37.     public JpaVendorAdapter jpaVendorAdapter() {
38.         HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new HibernateJpaVendorAdapter();
39.         hibernateJpaVendorAdapter.setShowSql(false);
40.         hibernateJpaVendorAdapter.setGenerateDdl(true);
41.         hibernateJpaVendorAdapter.setDatabase(Database.H2);
42.         return hibernateJpaVendorAdapter;
43.     }
44.
45.     // EntityManagerFactory
46.     @Bean

```

```

47.     public EntityManagerFactory entityManagerFactory(JpaVendorAdapter jpaVendorAdapter, DataSource
48.         dataSource) {
49.             LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();
50.             factory.setJpaVendorAdapter(jpaVendorAdapter);
51.             factory.setPackagesToScan("demo.entities");
52.             factory.setDataSource(dataSource);
53.             factory.afterPropertiesSet();
54.             return factory.getObject();
55.         }
56.     // Transaction manager
57.     @Bean
58.     public PlatformTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
59.         JpaTransactionManager txManager = new JpaTransactionManager();
60.         txManager.setEntityManagerFactory(entityManagerFactory);
61.         return txManager;
62.     }
63. }
64. }
```

- ligne 22 : l'annotation `[@Configuration]` fait de la classe `[Config]` une classe de configuration Spring ;
- ligne 21 : l'annotation `[@EnableJpaRepositories]` permet de désigner les dossiers où se trouvent les interfaces Spring Data `[CrudRepository]`. Ces interfaces vont devenir des composants Spring et être disponibles dans son contexte ;
- ligne 20 : l'annotation `[@EnableTransactionManagement]` indique que les méthodes des interfaces `[CrudRepository]` doivent se dérouler à l'intérieur d'une transaction ;
- ligne 19 : l'annotation `[@EntityScan]` permet de nommer les dossiers où doivent être cherchées les entités JPA. Ici elle a été mise en commentaires, parce que cette information a été donnée explicitement ligne 50. Cette annotation devrait être présente si on utilise le mode `[@EnableAutoConfiguration]` et que les entités JPA ne sont pas dans le même dossier que la classe de configuration ;
- ligne 18 : l'annotation `[@ComponentScan]` permet de lister les dossiers où les composants Spring doivent être recherchés. Les composants Spring sont des classes taguées avec des annotations Spring telles que `@Service`, `@Component`, `@Controller`, ... Ici il n'y en a pas d'autres que ceux qui sont définis au sein de la classe `[Config]`, aussi l'annotation a-t-elle été mise en commentaires ;
- lignes 25-33 : définissent la source de données, la base de données H2. C'est l'annotation `@Bean` de la ligne 25 qui fait de l'objet créé par cette méthode un composant géré par Spring. Le nom de la méthode peut être ici quelconque. Cependant elle doit être appelée `[dataSource]` si l'EntityManagerFactory de la ligne 47 est absent et défini par autoconfiguration ;
- ligne 29 : la base de données s'appellera `[demo]` et sera générée dans le dossier du projet ;
- lignes 36-43 : définissent l'implémentation JPA utilisée, ici une implémentation Hibernate. Le nom de la méthode peut être ici quelconque ;
- ligne 39 : pas de logs SQL ;
- ligne 30 : la base de données sera créée si elle n'existe pas ;
- lignes 46-54 : définissent l'EntityManagerFactory qui va gérer la persistance JPA. La méthode doit s'appeler obligatoirement `[entityManagerFactory]` ;
- ligne 47 : la méthode reçoit deux paramètres ayant le type des deux beans définis précédemment. Ceux-ci seront alors construits puis injectés par Spring comme paramètres de la méthode ;
- ligne 49 : fixe l'implémentation JPA utilisée ;
- ligne 50 : fixent les dossiers où trouver les entités JPA ;
- ligne 51 : fixe la source de donnée à gérer ;
- lignes 57-62 : le gestionnaire de transactions. La méthode doit s'appeler obligatoirement `[transactionManager]`. Elle reçoit pour paramètre le bean des lignes 46-54 ;
- ligne 60 : le gestionnaire de transactions est associé à l'EntityManagerFactory ;

Les méthodes précédentes peuvent être définies dans un ordre quelconque.

L'exécution du projet donne les mêmes résultats. Un nouveau fichier apparaît dans le dossier du projet, celui de la base de données H2 :

gs-accessing-data-jpa-2

```
  ▾ src/main/java
    ▾ Maven Dependencies
    ▾ JRE System Library [J2SE-1.5]
    ▾ dist
    ▾ gradle
    ▾ src
    ▾ target
      build.gradle
      demo.mv.db
      gradlew
      gradlew.bat
      pom.xml
```

Enfin, on peut se passer de Spring Boot. On crée une seconde classe exécutable [Main2] :

```
  ▾ src/main/java
    ▾ demo.config
      Config.java
    ▾ demo.console
      Main.java
      Main2.java
    ▾ demo.entities
      Customer.java
    ▾ demo.repositories
      CustomerRepository.java
```

La classe [Main2] a le code suivant :

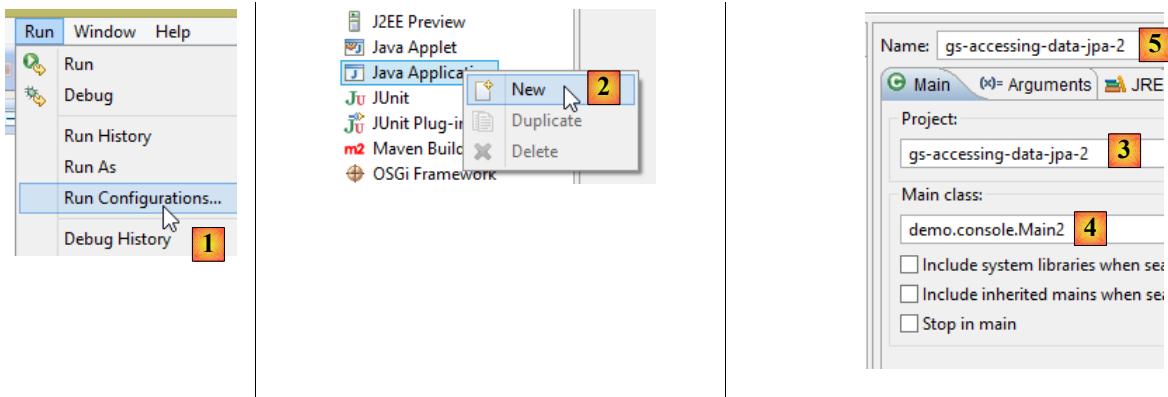
```
1. package demo.console;
2.
3. import java.util.List;
4.
5. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
6.
7. import demo.config.Config;
8. import demo.entities.Customer;
9. import demo.repositories.CustomerRepository;
10.
11. public class Main2 {
12.
13.     public static void main(String[] args) {
14.
15.         AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(Config.class);
16.         CustomerRepository repository = context.getBean(CustomerRepository.class);
17.     ....
18.
19.         context.close();
20.     }
21.
22. }
```

- ligne 15 : la classe de configuration [Config] est désormais exploitée par la classe Spring [AnnotationConfigApplicationContext]. On peut voir ligne 5 qu'il n'y a maintenant plus de dépendance vis à vis de Spring Boot.

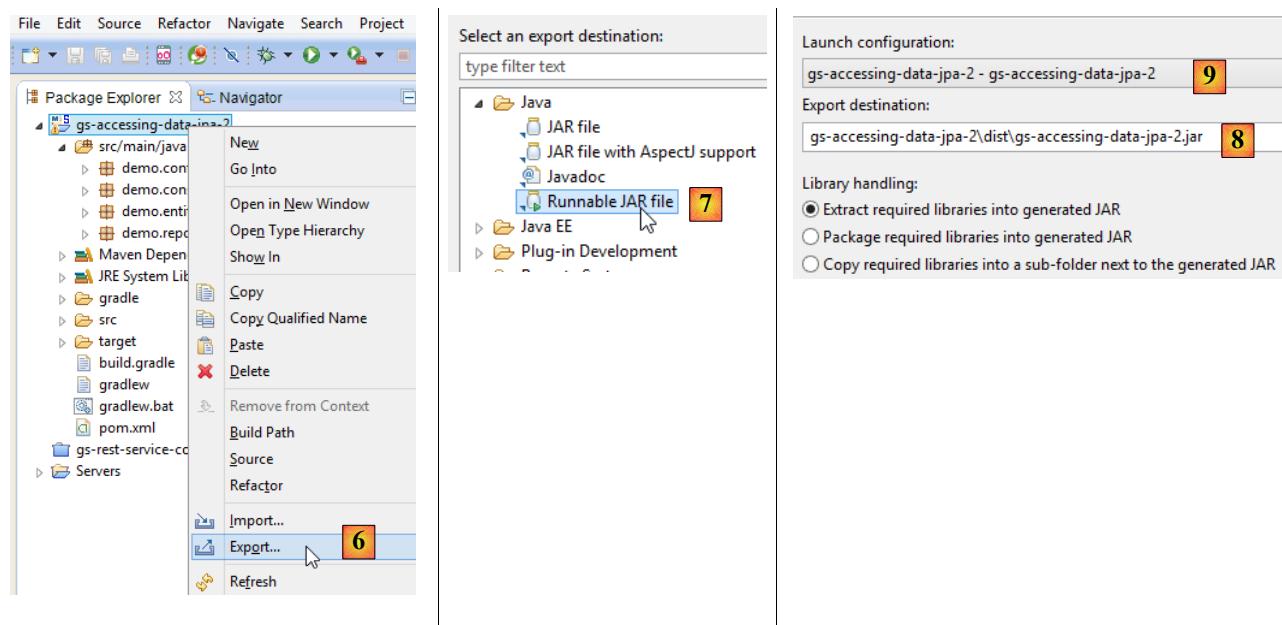
L'exécution donne les mêmes résultats que précédemment.

2.2.6 Crédation d'une archive exécutable

Pour créer une archive exécutable du projet, on peut procéder ainsi :



- en [1] : on crée une configuration d'exécution ;
- en [2] : de type [Java Application]
- en [3] : désigne le projet à exécuter (utiliser le bouton *Browse*) ;
- en [4] : désigne la classe à exécuter ;
- en [5] : le nom de la configuration d'exécution – peut être quelconque ;



- en [6] : on exporte le projet ;
- en [7] : sous la forme d'une archive JAR exécutable ;
- en [8] : indique le chemin et le nom du fichier exécutable à créer ;
- en [9] : le nom de la configuration d'exécution créée en [5] ;

Ceci fait, on ouvre une console dans le dossier contenant l'archive exécutable :

```
.....\dist>dir  
12/06/2014 09:11      15 104 869 gs-accessing-data-jpa-2.jar
```

L'archive est exécutée de la façon suivante :

-\dist>**java -jar gs-accessing-data-jpa-2.jar**

Les résultats obtenus dans la console sont les suivants :

- SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
- SLF4J: Defaulting to no-operation (NOP) logger implementation
- SLF4J: See <http://www.slf4j.org/codes.html#StaticLoggerBinder> for further details.

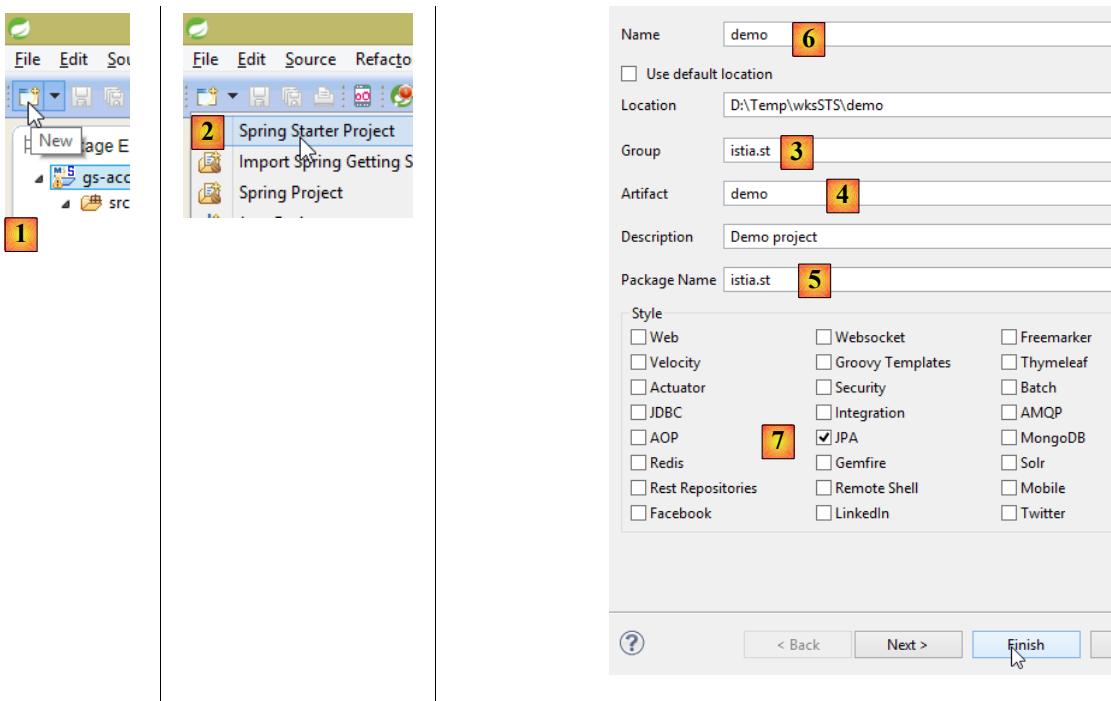
```

5. juin 12, 2014 9:48:38 AM org.hibernate.ejb.HibernatePersistence logDeprecation
6. WARN: HHH015016: Encountered a deprecated javax.persistence.spi.PersistenceProvider
   [org.hibernate.ejb.HibernatePersistence]; use [org.hibernate.jpa.HibernatePersistenceProvider] instead.
7. juin 12, 2014 9:48:38 AM org.hibernate.jpa.internal.util.LogHelper logPersistenceUnitInformation
8. INFO: HHH000204: Processing PersistenceUnitInfo [
9.       name: default
10.      ...
11. juin 12, 2014 9:48:38 AM org.hibernate.Version logVersion
12. INFO: HHH000412: Hibernate Core {4.3.4.Final}
13. juin 12, 2014 9:48:38 AM org.hibernate.cfg.Environment <clinit>
14. INFO: HHH000206: hibernate.properties not found
15. juin 12, 2014 9:48:38 AM org.hibernate.cfg.Environment buildBytecodeProvider
16. INFO: HHH00021: Bytecode provider name : javassist
17. juin 12, 2014 9:48:39 AM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
18. INFO: HCANN000001: Hibernate Commons Annotations {4.0.4.Final}
19. juin 12, 2014 9:48:39 AM org.hibernate.dialect.Dialect <init>
20. INFO: HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
21. juin 12, 2014 9:48:39 AM org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init>
22. INFO: HHH000397: Using ASTQueryTranslatorFactory
23. juin 12, 2014 9:48:40 AM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
24. INFO: HHH000228: Running hbm2ddl schema update
25. juin 12, 2014 9:48:40 AM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
26. INFO: HHH000102: Fetching database metadata
27. juin 12, 2014 9:48:40 AM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
28. INFO: HHH000396: Updating schema
29. juin 12, 2014 9:48:40 AM org.hibernate.tool.hbm2ddl.DatabaseMetadata getTableMetadata
30. INFO: HHH000262: Table not found: Customer
31. juin 12, 2014 9:48:40 AM org.hibernate.tool.hbm2ddl.DatabaseMetadata getTableMetadata
32. INFO: HHH000262: Table not found: Customer
33. juin 12, 2014 9:48:40 AM org.hibernate.tool.hbm2ddl.DatabaseMetadata getTableMetadata
34. INFO: HHH000262: Table not found: Customer
35. juin 12, 2014 9:48:40 AM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
36. INFO: HHH000232: Schema update complete
37. Customers found with findAll():
38. -----
39. Customer[id=1, firstName='Jack', lastName='Bauer']
40. Customer[id=2, firstName='Chloe', lastName='O'Brian']
41. Customer[id=3, firstName='Kim', lastName='Bauer']
42. Customer[id=4, firstName='David', lastName='Palmer']
43. Customer[id=5, firstName='Michelle', lastName='Dessler']
44.
45. Customer found with findOne(1L):
46. -----
47. Customer[id=1, firstName='Jack', lastName='Bauer']
48.
49. Customer found with findByLastName('Bauer'):
50. -----
51. Customer[id=1, firstName='Jack', lastName='Bauer']
52. Customer[id=3, firstName='Kim', lastName='Bauer']

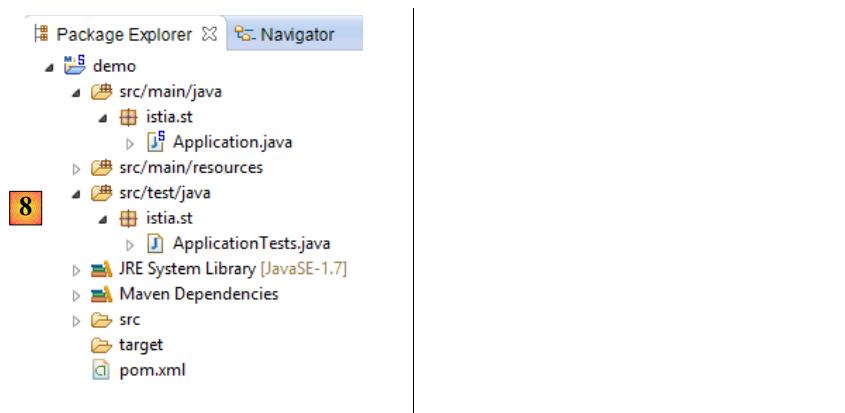
```

2.2.7 Créer un nouveau projet Spring Data

Pour créer un squelette de projet Spring Data, on peut procéder de la façon suivante :



- en [1], on crée un nouveau projet ;
- en [2] : de type [Spring Starter Project] ;
- le projet généré sera un projet Maven. En [3], on indique le nom du groupe du projet ;
- en [4] : on indique le nom de l'artifact (un jar ici) qui sera créé par construction du projet ;
- en [5] : on indique le package de la classe exécutable qui va être créée dans le projet ;
- en [6] : le nom Eclipse du projet – peut être quelconque (n'a pas à être identique à [4]) ;
- en [7] : on indique qu'on va créer un projet ayant une couche [JPA]. Les dépendances nécessaires à un tel projet vont alors être incluses dans le fichier [pom.xml] ;



- en [8] : le projet créé ;

Le fichier [pom.xml] intègre les dépendances nécessaires à un projet JPA :

```

1.   <parent>
2.     <groupId>org.springframework.boot</groupId>
3.     <artifactId>spring-boot-starter-parent</artifactId>
4.     <version>1.1.0.RELEASE</version>
5.     <relativePath/> <!-- lookup parent from repository -->
6.   </parent>
7.
8.   <dependencies>
9.     <dependency>
10.       <groupId>org.springframework.boot</groupId>

```

```

11.      <artifactId>spring-boot-starter-data-jpa</artifactId>
12.    </dependency>
13.    <dependency>
14.      <groupId>org.springframework.boot</groupId>
15.      <artifactId>spring-boot-starter-test</artifactId>
16.      <scope>test</scope>
17.    </dependency>
18.  </dependencies>

```

- lignes 9-12 : les dépendances nécessaires à JPA – vont inclure [Spring Data] ;
- lignes 13-17 : les dépendances nécessaires aux test JUnit intégrés avec Spring ;

La classe exécutable [Application] ne fait rien mais est pré-configurée :

```

1. package istia.st;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
5. import org.springframework.context.annotation.ComponentScan;
6. import org.springframework.context.annotation.Configuration;
7.
8. @Configuration
9. @ComponentScan
10. @EnableAutoConfiguration
11. public class Application {
12.
13.     public static void main(String[] args) {
14.         SpringApplication.run(Application.class, args);
15.     }
16. }

```

La classe de tests [ApplicationTests] ne fait rien mais est pré-configurée :

```

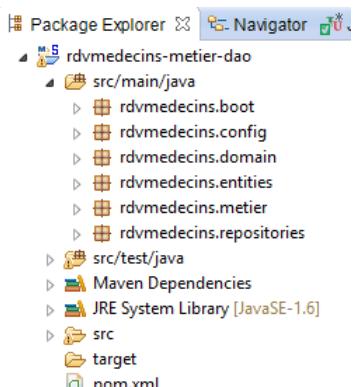
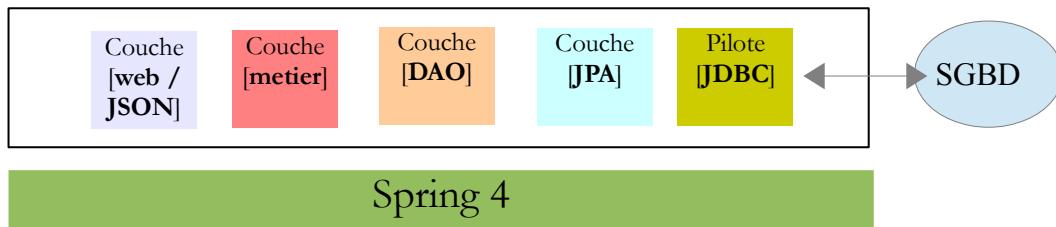
1. package istia.st;
2.
3. import org.junit.Test;
4. import org.junit.runner.RunWith;
5. import org.springframework.boot.test.SpringApplicationConfiguration;
6. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
7.
8. @RunWith(SpringJUnit4ClassRunner.class)
9. @SpringApplicationConfiguration(classes = Application.class)
10. public class ApplicationTests {
11.
12.     @Test
13.     public void contextLoads() {
14.     }
15.
16. }

```

- ligne 9 : l'annotation [@SpringApplicationConfiguration] permet d'exploiter le fichier de configuration [Application]. La classe de test bénéficiera ainsi de tous les beans qui seront définis par ce fichier ;
- ligne 8 : l'annotation [@RunWith] permet l'intégration de Spring avec JUnit : la classe va pouvoir être exécutée comme un test JUnit. [@RunWith] est une annotation JUnit (ligne 4) alors que la classe [SpringJUnit4ClassRunner] est une classe Spring (ligne 6) ;

Maintenant que nous avons un squelette d'application JPA, nous pouvons le compléter pour écrire le projet de la couche de persistance du serveur de notre application de gestion de rendez-vous.

2.3 Le projet Eclipse du serveur



Les éléments principaux du projet sont les suivants :

- [pom.xml] : fichier de configuration Maven du projet ;
- [rdvmedecins.entities] : les entités JPA ;
- [rdvmedecins.repositories] : les interfaces Spring Data d'accès aux entités JPA ;
- [rdvmedecins.metier] : la couche [métier] ;
- [rdvmedecins.domain] : les entités manipulées par la couche [métier] ;
- [rdvmedecins.config] : les classes de configuration de la couche de persistance ;
- [rdvmedecins.boot] : une application console basique ;

2.4 La configuration Maven

▶ Maven Dependencies	
▷ spring-boot-starter-data-jpa-1.1.0.RC1.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-boot-starter-data-jpa-1.1.0.RC1.jar	▷ hibernate-entitymanager-4.3.1.Final.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\hibernate-entitymanager-4.3.1.Final.jar
▷ spring-boot-starter-1.1.0.RC1.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-boot-starter-1.1.0.RC1.jar	▷ jboss-logging-3.1.3.GA.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\jboss-logging-3.1.3.GA.jar
▷ spring-boot-1.1.0.RC1.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-boot-1.1.0.RC1.jar	▷ jboss-logging-annotations-1.2.0.Beta1.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\jboss-logging-annotations-1.2.0.Beta1.jar
▷ spring-boot-autoconfigure-1.1.0.RC1.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-boot-autoconfigure-1.1.0.RC1.jar	▷ hibernate-core-4.3.1.Final.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\hibernate-core-4.3.1.Final.jar
▷ spring-boot-starter-logging-1.1.0.RC1.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-boot-starter-logging-1.1.0.RC1.jar	▷ antlr-2.7.7.jar - C:\Users\Serge Tahé\m2\repository\org\apache\antlr\antlr\2.7.7\antlr-2.7.7.jar
▷ jul-to-slf4j-1.7.7.jar - C:\Users\Serge Tahé\m2\repository\org\slf4j\jul-to-slf4j\1.7.7\jul-to-slf4j-1.7.7.jar	▷ jandex-1.1.0.Final.jar - C:\Users\Serge Tahé\m2\repository\com\codenameone\jandex\1.1.0\jandex-1.1.0.Final.jar
▷ log4j-over-slf4j-1.7.7.jar - C:\Users\Serge Tahé\m2\repository\org\log4j\log4j-over-slf4j\1.7.7\log4j-over-slf4j-1.7.7.jar	▷ dom4j-1.6.1.jar - C:\Users\Serge Tahé\m2\repository\org\xml\dom\dom4j\1.6.1\dom4j-1.6.1.jar
▷ logback-classic-1.1.2.jar - C:\Users\Serge Tahé\m2\repository\ch\qos\logback\logback-classic\1.1.2\logback-classic-1.1.2.jar	▷ xml-apis-1.0.b2.jar - C:\Users\Serge Tahé\m2\repository\xml-apis\xml-apis\1.0.b2\xml-apis-1.0.b2.jar
▷ logback-core-1.1.2.jar - C:\Users\Serge Tahé\m2\repository\ch\qos\logback\logback-core\1.1.2\logback-core-1.1.2.jar	▷ hibernate-commons-annotations-4.0.4.Final.jar - C:\Users\Serge Tahé\m2\repository\org\hibernate\hibernate-commons-annotations\4.0.4.Final\hibernate-commons-annotations-4.0.4.Final.jar
▷ snakeyaml-1.13.jar - C:\Users\Serge Tahé\m2\repository\org\yaml\snakeyaml\1.13\snakeyaml-1.13.jar	▷ hibernate-jpa-2.1-api-1.0.0.Final.jar - C:\Users\Serge Tahé\m2\repository\org\hibernate\hibernate-jpa-2.1-api\1.0.0\hibernate-jpa-2.1-api-1.0.0.Final.jar
▷ spring-boot-starter-aop-1.1.0.RC1.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-boot-starter-aop-1.1.0.RC1.jar	▷ jboss-transaction-api_1.2_spec-1.0.0.Final.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\jboss-transaction-api_1.2_spec-1.0.0.Final.jar
▷ spring-aop-4.0.5.RELEASE.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-aop-4.0.5.RELEASE.jar	▷ javassist-3.18.1-GA.jar - C:\Users\Serge Tahé\m2\repository\javassist\javassist\3.18.1-GA\javassist-3.18.1-GA.jar
▷ aopalliance-1.0.jar - C:\Users\Serge Tahé\m2\repository\org\codehaus\javassist\javassist\1.0\javassist-1.0.jar	▷ spring-core-4.0.5.RELEASE.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-core-4.0.5.RELEASE.jar
▷ aspectjrt-1.8.0.jar - C:\Users\Serge Tahé\m2\repository\org\aspectj\aspectjrt\1.8.0\aspectjrt-1.8.0.jar	▷ spring-orm-4.0.5.RELEASE.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-orm-4.0.5.RELEASE.jar
▷ aspectjweaver-1.8.0.jar - C:\Users\Serge Tahé\m2\repository\org\aspectj\aspectjweaver\1.8.0\aspectjweaver-1.8.0.jar	▷ spring-beans-4.0.5.RELEASE.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-beans-4.0.5.RELEASE.jar
▷ spring-boot-starter-jdbc-1.1.0.RC1.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-boot-starter-jdbc-1.1.0.RC1.jar	▷ spring-data-jpa-1.6.0.RELEASE.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-data-jpa-1.6.0.RELEASE.jar
▷ spring-jdbc-4.0.5.RELEASE.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-jdbc-4.0.5.RELEASE.jar	▷ spring-data-commons-1.8.0.RELEASE.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-data-commons-1.8.0.RELEASE.jar
▷ tomcat-jdbc-7.0.54.jar - C:\Users\Serge Tahé\m2\repository\org\apache\tomcat\tomcat-jdbc\7.0.54\tomcat-jdbc-7.0.54.jar	▷ spring-context-4.0.5.RELEASE.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-context-4.0.5.RELEASE.jar
▷ tomcat-juli-7.0.54.jar - C:\Users\Serge Tahé\m2\repository\org\apache\tomcat\tomcat-juli\7.0.54\tomcat-juli-7.0.54.jar	▷ spring-expression-4.0.5.RELEASE.jar - C:\Users\Serge Tahé\IdeaProjects\rdvmedecins\target\classes\spring-expression-4.0.5.RELEASE.jar
▷ spring-tx-4.0.5.RELEASE.jar - C:\Users\Serge Tahé\m2\repository\org\springframework\tx\spring-tx\4.0.5\spring-tx-4.0.5.RELEASE.jar	

Le fichier [pom.xml] du projet est le suivant :

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xsi:schemaLocation="http://maven.apache.org/maven-v4_0_0.xsd"
   http://maven.apache.org/POM/4.0.0
3.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4.   <modelVersion>4.0.0</modelVersion>
5.   <groupId>istia.st.spring4.rdvmedecins</groupId>
6.   <artifactId>rdvmedecins-metier-dao</artifactId>
7.   <version>0.0.1-SNAPSHOT</version>
8.   <parent>
9.     <groupId>org.springframework.boot</groupId>
10.    <artifactId>spring-boot-starter-parent</artifactId>
11.    <version>1.0.0.RELEASE</version>
12.  </parent>
13.  <dependencies>
14.    <dependency>
15.      <groupId>org.springframework.boot</groupId>
16.      <artifactId>spring-boot-starter-data-jpa</artifactId>
17.    </dependency>
18.    <dependency>
19.      <groupId>org.springframework.boot</groupId>
20.      <artifactId>spring-boot-starter-test</artifactId>
21.      <scope>test</scope>
22.    </dependency>
23.    <dependency>
24.      <groupId>mysql</groupId>
25.      <artifactId>mysql-connector-java</artifactId>
26.    </dependency>
27.    <dependency>
28.      <groupId>commons-dbcp</groupId>
29.      <artifactId>commons-dbcp</artifactId>
30.    </dependency>
31.    <dependency>
32.      <groupId>commons-pool</groupId>
33.      <artifactId>commons-pool</artifactId>
34.    </dependency>
35.    <dependency>
36.      <groupId>com.fasterxml.jackson.core</groupId>
37.      <artifactId>jackson-databind</artifactId>
38.    </dependency>
39.    <dependency>
40.      <groupId>com.google.guava</groupId>
41.      <artifactId>guava</artifactId>
42.      <version>16.0.1</version>
43.    </dependency>
44.  </dependencies>
45.  <properties>
46.    <!-- use UTF-8 for everything -->
47.    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```

48.      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
49.      <start-class>istia.st.spring.data.main.Application</start-class>
50.  </properties>
51.  <build>
52.      <plugins>
53.          <plugin>
54.              <artifactId>maven-compiler-plugin</artifactId>
55.          </plugin>
56.          <plugin>
57.              <groupId>org.springframework.boot</groupId>
58.              <artifactId>spring-boot-maven-plugin</artifactId>
59.          </plugin>
60.      </plugins>
61.  </build>
62.  <repositories>
63.      <repository>
64.          <id>spring-milestones</id>
65.          <name>Spring Milestones</name>
66.          <url>http://repo.spring.io/libs-milestone</url>
67.          <snapshots>
68.              <enabled>false</enabled>
69.          </snapshots>
70.      </repository>
71.      <repository>
72.          <id>org.jboss.repository.releases</id>
73.          <name>JBoss Maven Release Repository</name>
74.          <url>https://repository.jboss.org/nexus/content/repositories/releases</url>
75.          <snapshots>
76.              <enabled>false</enabled>
77.          </snapshots>
78.      </repository>
79.  </repositories>
80.  <pluginRepositories>
81.      <pluginRepository>
82.          <id>spring-milestones</id>
83.          <name>Spring Milestones</name>
84.          <url>http://repo.spring.io/libs-milestone</url>
85.          <snapshots>
86.              <enabled>false</enabled>
87.          </snapshots>
88.      </pluginRepository>
89.  </pluginRepositories>
90. </project>

```

- lignes 8-12 : le projet s'appuie sur le projet parent [spring-boot-starter-parent]. Pour les dépendances déjà présentes dans le projet parent, on ne précise pas de version. C'est la version définie dans le parent qui sera utilisée. Pour les autres dépendances, on les déclare normalement ;
- lignes 14-17 : pour Spring Data ;
- lignes 18-22 : pour les tests JUnit ;
- lignes 23-26 : pilote JDBC du SGBD MySQL5 ;
- lignes 27-34 : pool de connexions Commons DBCP ;
- lignes 35-38 : bibliothèque Jackson de gestion du JSON ;
- lignes 39-43 : bibliothèque Google de gestion des collections ;

La version 1.1.0.RC1 de [spring-boot-starter-parent] utilise les versions suivantes des bibliothèques :

```

1.  <activemq.version>5.9.1</activemq.version>
2.  <aspectj.version>1.8.0</aspectj.version>
3.  <codahale-metrics.version>3.0.2</codahale-metrics.version>
4.  <commons-beanutils.version>1.9.1</commons-beanutils.version>
5.  <commons-collections.version>3.2.1</commons-collections.version>
6.  <commons-dbcp.version>1.4</commons-dbcp.version>
7.  <commons-digester.version>2.1</commons-digester.version>
8.  <commons-pool.version>1.6</commons-pool.version>
9.  <commons-pool2.version>2.2</commons-pool2.version>
10. <crashub.version>1.3.0-beta20</crashub.version>
11. <flyway.version>3.0</flyway.version>
12. <freemarker.version>2.3.20</freemarker.version>
13. <gemfire.version>7.0.2</gemfire.version>
14. <gradle.version>1.6</gradle.version>
15. <groovy.version>2.3.2</groovy.version>
16. <h2.version>1.3.175</h2.version>

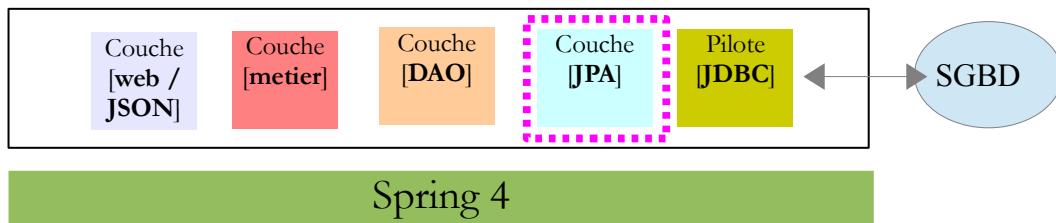
```

```

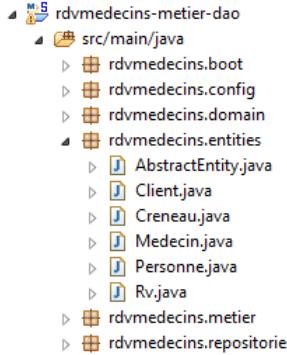
17. <hamcrest.version>1.3</hamcrest.version>
18. <hibernate-entitymanager.version>4.3.1.Final</hibernate-entitymanager.version>
19. <hibernate-jpa-api.version>1.0.1.Final</hibernate-jpa-api.version>
20. <hibernate-validator.version>5.0.3.Final</hibernate-validator.version>
21. <hibernate.version>4.3.1.Final</hibernate.version>
22. <hikaricp.version>1.3.8</hikaricp.version>
23. <hornetq.version>2.4.1.Final</hornetq.version>
24. <hsqldb.version>2.3.2</hsqldb.version>
25. <httpasyncclient.version>4.0.1</httpasyncclient.version>
26. <httpclient.version>4.3.3</httpclient.version>
27. <jackson.version>2.3.3</jackson.version>
28. <java.version>1.6</java.version>
29. <javassist.version>3.18.1-GA</javassist.version>
30. <jedis.version>2.4.1</jedis.version>
31. <jetty-jsp.version>2.2.0.v201112011158</jetty-jsp.version>
32. <jetty.version>8.1.14.v20131031</jetty.version>
33. <joda-time.version>2.3</joda-time.version>
34. <jolokia.version>1.2.0</jolokia.version>
35. <jstl.version>1.2</jstl.version>
36. <junit.version>4.11</junit.version>
37. <liquibase.version>3.0.8</liquibase.version>
38. <log4j.version>1.2.17</log4j.version>
39. <logback.version>1.1.2</logback.version>
40. <mockito.version>1.9.5</mockito.version>
41. <mongodb.version>2.12.1</mongodb.version>
42. <mysql.version>5.1.30</mysql.version>
43. <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
44. <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
45. <reactor.version>1.1.1.RELEASE</reactor.version>
46. <servlet-api.version>3.0.1</servlet-api.version>
47. <slf4j.version>1.7.7</slf4j.version>
48. <snakeyaml.version>1.13</snakeyaml.version>
49. <solr.version>4.7.2</solr.version>
50. <spock.version>0.7-groovy-2.0</spock.version>
51. <spring-amqp.version>1.3.4.RELEASE</spring-amqp.version>
52. <spring-batch.version>3.0.0.RELEASE</spring-batch.version>
53. <spring-boot.version>1.1.0.RC1</spring-boot.version>
54. <spring-data-releasetrain.version>Dijkstra-RELEASE</spring-data-releasetrain.version>
55. <spring-hateoas.version>0.12.0.RELEASE</spring-hateoas.version>
56. <spring-integration.version>4.0.2.RELEASE</spring-integration.version>
57. <spring-loaded.version>1.2.0.RELEASE</spring-loaded.version>
58. <spring-mobile.version>1.1.1.RELEASE</spring-mobile.version>
59. <spring-security-jwt.version>1.0.2.RELEASE</spring-security-jwt.version>
60. <spring-security.version>3.2.4.RELEASE</spring-security.version>
61. <spring-social-facebook.version>1.1.1.RELEASE</spring-social-facebook.version>
62. <spring-social-linkedin.version>1.0.1.RELEASE</spring-social-linkedin.version>
63. <spring-social-twitter.version>1.1.0.RELEASE</spring-social-twitter.version>
64. <spring-social.version>1.1.0.RELEASE</spring-social.version>
65. <spring.version>4.0.5.RELEASE</spring.version>
66. <thymeleaf-extras-springsecurity3.version>2.1.1.RELEASE</thymeleaf-extras-springsecurity3.version>
67. <thymeleaf-layout-dialect.version>1.2.4</thymeleaf-layout-dialect.version>
68. <thymeleaf.version>2.1.3.RELEASE</thymeleaf.version>
69. <tomcat.version>7.0.54</tomcat.version>
70. <velocity-tools.version>2.0</velocity-tools.version>
71. <velocity.version>1.7</velocity.version>

```

2.5 Les entités JPA



Les entités JPA sont les objets qui vont encapsuler les lignes des tables de la base de données.



La classe [AbstractEntity] est la classe parent des entités [Personne, Creneau, Rv]. Sa définition est la suivante :

```

1. package rdvmedecins.entities;
2.
3. import java.io.Serializable;
4.
5. import javax.persistence.GeneratedValue;
6. import javax.persistence.GenerationType;
7. import javax.persistence.Id;
8. import javax.persistence.MappedSuperclass;
9. import javax.persistence.Version;
10.
11. @MappedSuperclass
12. public class AbstractEntity implements Serializable {
13.
14.     private static final long serialVersionUID = 1L;
15.     @Id
16.     @GeneratedValue(strategy = GenerationType.AUTO)
17.     protected Long id;
18.     @Version
19.     protected Long version;
20.
21.     @Override
22.     public int hashCode() {
23.         int hash = 0;
24.         hash += (id != null ? id.hashCode() : 0);
25.         return hash;
26.     }
27.
28.     // initialisation
29.     public AbstractEntity build(Long id, Long version) {
30.         this.id = id;
31.         this.version = version;
32.         return this;
33.     }
34.
35.     @Override
36.     public boolean equals(Object entity) {
37.         String class1 = this.getClass().getName();
38.         String class2 = entity.getClass().getName();
39.         if (!class2.equals(class1)) {
40.             return false;
41.         }
42.         AbstractEntity other = (AbstractEntity) entity;
43.         return this.id == other.id;
44.     }
45.
46.     // getters et setters
47.     ..
48. }
```

- ligne 11 : l'annotation [`@MappedSuperclass`] indique que la classe annotée est parente d'entités JPA [`@Entity`] ;
- lignes 15-17 : définissent la clé primaire [`id`] de chaque entité. C'est l'annotation [`@Id`] qui fait du champ [`id`] une clé primaire. L'annotation [`@GeneratedValue(strategy = GenerationType.AUTO)`] indique que la valeur de cette clé primaire est générée par le SGBD et qu'aucun mode de génération n'est imposé ;

- lignes 18-19 : définissent la version de chaque entité. L'implémentation JPA va incrémenter ce n° de version à chaque fois que l'entité sera modifiée. Ce n° sert à empêcher la mise à jour simultanée de l'entité par deux utilisateurs différents : deux utilisateurs U1 et U2 lisent l'entité E avec un n° de version égal à V1. U1 modifie E et persiste cette modification en base : le n° de version passe alors à V1+1. U2 modifie E à son tour et persiste cette modification en base : il recevra une exception car il possède une version (V1) différente de celle en base (V1+1) ;
- lignes 29-33 : la méthode [build] permet d'initialiser les deux champs de [AbstractEntity]. Cette méthode rend la référence de l'instance [AbstractEntity] ainsi initialisée ;
- lignes 36-44 : la méthode [equals] de la classe est redéfinie : deux entités seront dites égales si elles ont le même nom de classe et le même identifiant **id** ;

L'entité [Personne] est la classe parente des entités [Medecin] et [Client] :

```

1. package rdvmedecins.entities;
2.
3. import javax.persistence.Column;
4. import javax.persistence.MappedSuperclass;
5.
6. @MappedSuperclass
7. public class Personne extends AbstractEntity {
8.     private static final long serialVersionUID = 1L;
9.     // attributs d'une personne
10.    @Column(length = 5)
11.    private String titre;
12.    @Column(length = 20)
13.    private String nom;
14.    @Column(length = 20)
15.    private String prenom;
16.
17.    // constructeur par défaut
18.    public Personne() {
19.    }
20.
21.    // constructeur avec paramètres
22.    public Personne(String titre, String nom, String prenom) {
23.        this.titre = titre;
24.        this.nom = nom;
25.        this.prenom = prenom;
26.    }
27.
28.    // toString
29.    public String toString() {
30.        return String.format("Personne[%s, %s, %s, %s, %s]", id, version, titre, nom, prenom);
31.    }
32.
33.    // getters et setters
34.    ...
35. }
```

- ligne 6 : l'annotation [@MappedSuperclass] indique que la classe annotée est parente d'entités JPA [@Entity] ;
- lignes 10-15 : une personne a un titre (Melle), un prénom (Jacqueline), un nom (Tatou). Aucune information n'est donnée sur les colonnes de la table. Elles porteront donc par défaut les mêmes noms que les champs ;

L'entité [Medecin] est la suivante :

```

1. package rdvmedecins.entities;
2.
3. import javax.persistence.Entity;
4. import javax.persistence.Table;
5.
6. @Entity
7. @Table(name = "medecins")
8. public class Medecin extends Personne {
9.
10.    private static final long serialVersionUID = 1L;
11.
12.    // constructeur par défaut
13.    public Medecin() {
14.    }
15.
16.    // constructeur avec paramètres
17.    public Medecin(String titre, String nom, String prenom) {
```

```

18.     super(titre, nom, prenom);
19. }
20.
21. public String toString() {
22.     return String.format("Medecin[%s]", super.toString());
23. }
24.
25. }

```

- ligne 6 : la classe est une entité JPA ;
- ligne 7 : associée à la table [MEDECINS] de la base de données ;
- ligne 8 : l'entité [Medecin] dérive de l'entité [Personne] ;

Un médecin pourra être initialisé de la façon suivante :

```
Medecin m=new Medecin("Mr","Paul","Tatou");
```

Si de plus, on veut lui affecter un identifiant et une version on pourra écrire :

```
Medecin m=new Medecin("Mr","Paul","Tatou").build(10,1);
```

où la méthode [build] est celle définie dans [AbstractEntity].

L'entité [Client] est la suivante :

```

1. package rdvmedecins.entities;
2.
3. import javax.persistence.Entity;
4. import javax.persistence.Table;
5.
6. @Entity
7. @Table(name = "clients")
8. public class Client extends Personne {
9.
10.    private static final long serialVersionUID = 1L;
11.
12.    // constructeur par défaut
13.    public Client() {
14.    }
15.
16.    // constructeur avec paramètres
17.    public Client(String titre, String nom, String prenom) {
18.        super(titre, nom, prenom);
19.    }
20.
21.    // identité
22.    public String toString() {
23.        return String.format("Client[%s]", super.toString());
24.    }
25.
26. }

```

- ligne 6 : la classe est une entité JPA ;
- ligne 7 : associée à la table [CLIENTS] de la base de données ;
- ligne 8 : l'entité [Client] dérive de l'entité [Personne] ;

L'entité [Creneau] est la suivante :

```

1. package rdvmedecins.entities;
2.
3. import javax.persistence.Column;
4. import javax.persistence.Entity;
5. import javax.persistence.FetchType;
6. import javax.persistence.JoinColumn;
7. import javax.persistence.ManyToOne;
8. import javax.persistence.Table;
9.
10. @Entity
11. @Table(name = "creneaux")
12. public class Creneau extends AbstractEntity {

```

```

13.
14.     private static final long serialVersionUID = 1L;
15.     // caractéristiques d'un créneau de RV
16.     private int hdebut;
17.     private int mdebut;
18.     private int hfin;
19.     private int mfin;
20.
21.     // un créneau est lié à un médecin
22.     @ManyToOne(fetch = FetchType.LAZY)
23.     @JoinColumn(name = "id_medecin")
24.     private Medecin medecin;
25.
26.     // clé étrangère
27.     @Column(name = "id_medecin", insertable = false, updatable = false)
28.     private long idMedecin;
29.
30.     // constructeur par défaut
31.     public Creneau() {
32.     }
33.
34.     // constructeur avec paramètres
35.     public Creneau(Medecin medecin, int hdebut, int mdebut, int hfin, int mfin) {
36.         this.medecin = medecin;
37.         this.hdebut = hdebut;
38.         this.mdebut = mdebut;
39.         this.hfin = hfin;
40.         this.mfin = mfin;
41.     }
42.
43.     // toString
44.     public String toString() {
45.         return String.format("Créneau[%d, %d, %d, %d:%d, %d:%d]", id, version, idMedecin, hdebut, mdebut,
46.         hfin, mfin);
47.     }
48.     // clé étrangère
49.     public long getIdMedecin() {
50.         return idMedecin;
51.     }
52.
53.     // setters - getters
54.     ...
55. }
```

- ligne 10 : la classe est une entité JPA ;
- ligne 11 : associée à la table [CRENEAUX] de la base de données ;
- ligne 12 : l'entité [Creneau] dérive de l'entité [AbstractEntity] et hérite donc de l'identifiant [id] et de la version [version] ;
- ligne 16 : heure de début du créneau (14) ;
- ligne 17 : minutes de début du créneau (20) ;
- ligne 18 : heure de fin du créneau (14) ;
- ligne 19 : minutes de fin du créneau (40) ;
- lignes 22-24 : le médecin propriétaire du créneau. La table [CRENEAUX] a une clé étrangère sur la table [MEDECINS]. Cette relation est matérialisée par les lignes 22-24 ;
- ligne 22 : l'annotation `[@ManyToOne]` signale une relation **plusieurs** (créneaux) à **un** (médecin). L'attribut `[fetch=FetchType.LAZY]` indique que lorsqu'on demande une entité [Creneau] au contexte de persistance et que celle-ci doit être cherchée dans la base de données, alors l'entité [Medecin] n'est pas ramenée avec elle. L'intérêt de ce mode est que l'entité [Medecin] n'est cherchée que si le développeur le demande. On économise ainsi la mémoire et on gagne en performances ;
- ligne 23 : indique le nom de la colonne clé étrangère dans la table [CRENEAUX] ;
- lignes 27-28 : la clé étrangère sur la table [MEDECINS] ;
- ligne 27 : la colonne [ID_MEDECIN] a déjà été utilisée ligne 23. Cela veut dire qu'elle peut être modifiée par deux voies différentes ce que n'accepte pas la norme JPA. On ajoute donc les attributs `[insertable = false, updatable = false]`, ce qui fait que la colonne ne peut qu'être lue ;

L'entité [Rv] est la suivante :

```

1. package rdvmedecins.entities;
2.
3. import java.util.Date;
```

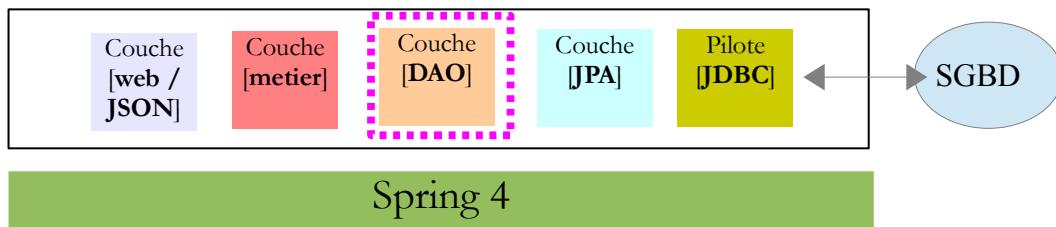
```

4.
5. import javax.persistence.Column;
6. import javax.persistence.Entity;
7. import javax.persistence.FetchType;
8. import javax.persistence.JoinColumn;
9. import javax.persistence.ManyToOne;
10. import javax.persistence.Table;
11. import javax.persistence.Temporal;
12. import javax.persistence.TemporalType;
13.
14. @Entity
15. @Table(name = "rv")
16. public class Rv extends AbstractEntity {
17.     private static final long serialVersionUID = 1L;
18.
19.     // caractéristiques d'un Rv
20.     @Temporal(TemporalType.DATE)
21.     private Date jour;
22.
23.     // un rv est lié à un client
24.     @ManyToOne(fetch = FetchType.LAZY)
25.     @JoinColumn(name = "id_client")
26.     private Client client;
27.
28.     // un rv est lié à un créneau
29.     @ManyToOne(fetch = FetchType.LAZY)
30.     @JoinColumn(name = "id_creneau")
31.     private Creneau creneau;
32.
33.     // clés étrangères
34.     @Column(name = "id_client", insertable = false, updatable = false)
35.     private long idClient;
36.     @Column(name = "id_creneau", insertable = false, updatable = false)
37.     private long idCreneau;
38.
39.     // constructeur par défaut
40.     public Rv() {
41.     }
42.
43.     // avec paramètres
44.     public Rv(Date jour, Client client, Creneau creneau) {
45.         this.jour = jour;
46.         this.client = client;
47.         this.creneau = creneau;
48.     }
49.
50.     // toString
51.     public String toString() {
52.         return String.format("Rv[%d, %s, %d, %d]", id, jour, client.id, creneau.id);
53.     }
54.
55.     // clés étrangères
56.     public long getIdCreneau() {
57.         return idCreneau;
58.     }
59.
60.     public long getIdClient() {
61.         return idClient;
62.     }
63.
64.     // getters et setters
65. ...
66. }
```

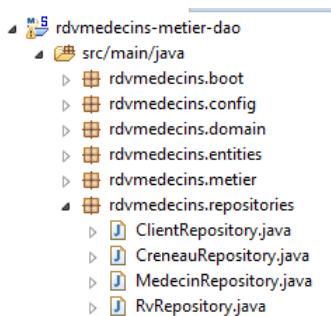
- ligne 14 : la classe est une entité JPA ;
- ligne 15 : associée à la table [RV] de la base de données ;
- ligne 16 : l'entité [Rv] dérive de l'entité [AbstractEntity] et hérite donc de l'identifiant [id] et de la version [version] ;
- ligne 21 : la date du rendez-vous ;
- ligne 20 : le type [Date] de Java contient à la fois une date et une heure. Ici on précise que seule la date est utilisée ;
- lignes 24-26 : le client pour lequel ce rendez-vous a été pris. La table [RV] a une clé étrangère sur la table [CLIENTS]. Cette relation est matérialisée par les lignes 24-26 ;
- lignes 29-31 : le créneau horaire du rendez-vous. La table [RV] a une clé étrangère sur la table [CRENEAUX]. Cette relation est matérialisée par les lignes 29-31 ;

- lignes 34-35 : la clé étrangère [idClient] ;
- lignes 36-37 : la clé étrangère [idCreneau] ;

2.6 La couche [DAO]



Nous allons implémenter la couche [DAO] avec Spring Data :



La couche [DAO] est implémentée avec quatre interfaces Spring Data :

- [ClientRepository] : donne accès aux entités JPA [Client] ;
- [CreneauRepository] : donne accès aux entités JPA [Creneau] ;
- [MedecinRepository] : donne accès aux entités JPA [Medecin] ;
- [RvRepository] : donne accès aux entités JPA [Rv] ;

L'interface [MedecinRepository] est la suivante :

```

1. package rdvmedecins.repositories;
2.
3. import org.springframework.data.repository.CrudRepository;
4.
5. import rdvmedecins.entities.Medecin;
6.
7. public interface MedecinRepository extends CrudRepository<Medecin, Long> {
8. }
```

- ligne 7 : l'interface [MedecinRepository] se contente d'hériter des méthodes de l'interface [CrudRepository] sans en ajouter d'autres ;

L'interface [ClientRepository] est la suivante :

```

1. package rdvmedecins.repositories;
2.
3. import org.springframework.data.repository.CrudRepository;
4.
5. import rdvmedecins.entities.Client;
6.
7. public interface ClientRepository extends CrudRepository<Client, Long> {
8. }
```

- ligne 7 : l'interface [ClientRepository] se contente d'hériter des méthodes de l'interface [CrudRepository] sans en ajouter d'autres ;

L'interface [**CreneauRepository**] est la suivante :

```

1. package rdvmedecins.repositories;
2.
3. import org.springframework.data.jpa.repository.Query;
4. import org.springframework.data.repository.CrudRepository;
5.
6. import rdvmedecins.entities.Creneau;
7.
8. public interface CreneauRepository extends CrudRepository<Creneau, Long> {
9.     // liste des créneaux horaires d'un médecin
10.    @Query("select c from Creneau c where c.medecin.id=?1")
11.    Iterable<Creneau> getAllCreneaux(long idMedecin);
12. }
```

- ligne 8 : l'interface [CreneauRepository] hérite des méthodes de l'interface [CrudRepository] ;
- lignes 10-11 : la méthode [getAllCreneaux] permet d'avoir les créneaux horaires d'un médecin ;
- ligne 11 : le paramètre est l'identifiant du médecin. Le résultat est une liste de créneaux horaires sous la forme d'un objet [Iterable<Creneau>] ;
- ligne 10 : l'annotation [@Query] permet de spécifier la requête JPQL (Java Persistence Query Language) qui implémente la méthode. Le paramètre [?1] sera remplacé par le paramètre [idMedecin] de la méthode ;

L'interface [**RvRepository**] est la suivante :

```

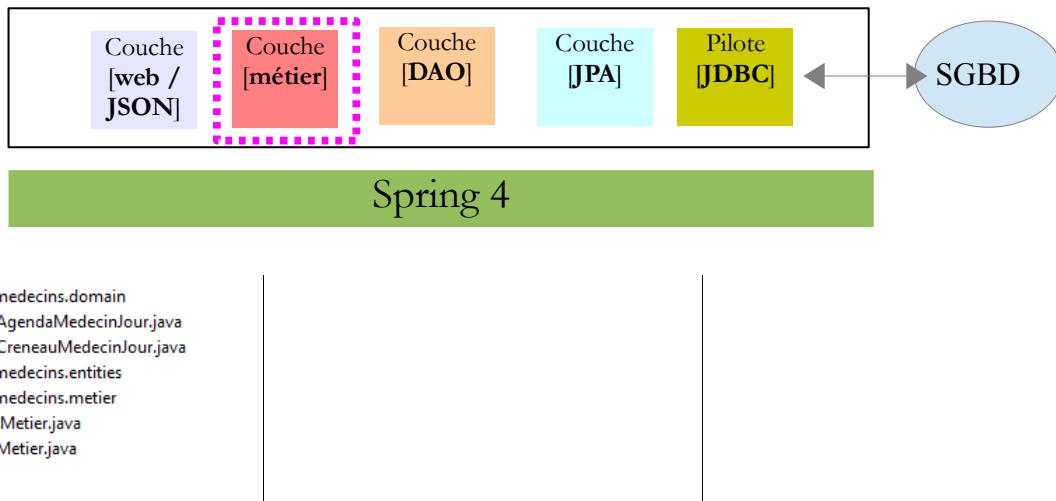
1. package rdvmedecins.repositories;
2.
3. import java.util.Date;
4.
5. import org.springframework.data.jpa.repository.Query;
6. import org.springframework.data.repository.CrudRepository;
7.
8. import rdvmedecins.entities.Rv;
9.
10. public interface RvRepository extends CrudRepository<Rv, Long> {
11.
12.     @Query("select rv from Rv rv left join fetch rv.client c left join fetch rv.creneau cr where
13.             cr.medecin.id=?1 and rv.jour=?2")
14.     Iterable<Rv> getRvMedecinJour(long idMedecin, Date jour);
```

- ligne 10 : l'interface [RvRepository] hérite des méthodes de l'interface [CrudRepository] ;
- lignes 12-13 : la méthode [getRvMedecinJour] permet d'avoir les rendez-vous d'un médecin pour un jour donné ;
- ligne 13 : les paramètres sont l'identifiant du médecin et le jour. Le résultat est une liste de rendez-vous sous la forme d'un objet [Iterable<Rv>] ;
- ligne 12 : l'annotation [@Query] permet de spécifier la requête JPQL qui implémente la méthode. Le paramètre [?1] sera remplacé par le paramètre [idMedecin] de la méthode et le paramètre [?2] sera remplacé par le paramètre [jour] de la méthode. On ne peut se contenter de la requête JPQL suivante :

```
select rv from Rv rv where rv.creneau.medecin.id=?1 and rv.jour=?2
```

car les champs de la classe Rv, de types [Client] et [Creneau] sont obtenus en mode [FetchType.LAZY], ce qui signifie qu'ils doivent être demandés explicitement pour être obtenus. Ceci est fait dans la requête JPQL avec la syntaxe [left join fetch entité] qui demande qu'une jointure soit faite avec la table sur laquelle pointe la clé étrangère afin de récupérer l'entité pointée ;

2.7 La couche [métier]



- [IMetier] est l'interface de la couche [métier] et [Metier] son implémentation ;
- [AgendaMedecinJour] et [CreneauMedecinJour] sont deux entités métier ;

2.7.1 Les entités

L'entité [CreneauMedecinJour] associe un créneau horaire et le rendez-vous éventuel pris dans ce créneau :

```
1. package rdvmedecins.domain;
2.
3. import java.io.Serializable;
4.
5. import rdvmedecins.entities.Creneau;
6. import rdvmedecins.entities.Rv;
7.
8. public class CreneauMedecinJour implements Serializable {
9.
10.    private static final long serialVersionUID = 1L;
11.    // champs
12.    private Creneau creneau;
13.    private Rv rv;
14.
15.    // constructeurs
16.    public CreneauMedecinJour() {
17.
18.    }
19.
20.    public CreneauMedecinJour(Creneau creneau, Rv rv) {
21.        this.creneau=creneau;
22.        this.rv=rv;
23.    }
24.
25.    // toString
26.    @Override
27.    public String toString() {
28.        return String.format("[%s %s]", creneau, rv);
29.    }
30.
31.    // getters et setters
32.    ...
33. }
```

- ligne 12 : le créneau horaire ;
- ligne 13 : l'éventuel rendez-vous – *null* sinon ;

L'entité [AgendaMedecinJour] est l'agenda d'un médecin pour un jour donné, c-à-d la liste de ses rendez-vous :

```

1. package rdvmedecins.domain;
2.
3. import java.io.Serializable;
4. import java.text.SimpleDateFormat;
5. import java.util.Date;
6.
7. import rdvmedecins.entities.Medecin;
8.
9. public class AgendaMedecinJour implements Serializable {
10.
11.     private static final long serialVersionUID = 1L;
12.     // champs
13.     private Medecin medecin;
14.     private Date jour;
15.     private CreneauMedecinJour[] creneauxMedecinJour;
16.
17.     // constructeurs
18.     public AgendaMedecinJour() {
19.
20. }
21.
22.     public AgendaMedecinJour(Medecin medecin, Date jour, CreneauMedecinJour[] creneauxMedecinJour) {
23.         this.medecin = medecin;
24.         this.jour = jour;
25.         this.creneauxMedecinJour = creneauxMedecinJour;
26.     }
27.
28.     public String toString() {
29.         StringBuffer str = new StringBuffer("");
30.         for (CreneauMedecinJour cr : creneauxMedecinJour) {
31.             str.append(" ");
32.             str.append(cr.toString());
33.         }
34.         return String.format("Agenda[%s,%s,%s]", medecin, new SimpleDateFormat("dd/MM/yyyy").format(jour),
35.             str.toString());
36.     }
37.     // getters et setters
38.     ...
39. }
```

- ligne 13 : le médecin ;
- ligne 14 : le jour dans l'agenda ;
- ligne 15 : ses créneaux horaires avec ou sans rendez-vous ;

2.7.2 Le service

L'interface de la couche [métier] est la suivante :

```

1. package rdvmedecins.metier;
2.
3. import java.util.Date;
4. import java.util.List;
5.
6. import rdvmedecins.domain.AgendaMedecinJour;
7. import rdvmedecins.entities.Client;
8. import rdvmedecins.entities.Creneau;
9. import rdvmedecins.entities.Medecin;
10. import rdvmedecins.entities.Rv;
11.
12. public interface IMetier {
13.
14.     // liste des clients
15.     public List<Client> getAllClients();
16.
17.     // liste des Médecins
18.     public List<Medecin> getAllMedecins();
19.
20.     // liste des créneaux horaires d'un médecin
21.     public List<Creneau> getAllCreneaux(long idMedecin);
22.
23.     // liste des Rv d'un médecin, un jour donné
```

```

24.     public List<Rv> getRvMedecinJour(long idMedecin, Date jour);
25.
26.     // trouver un client identifié par son id
27.     public Client getClientById(long id);
28.
29.     // trouver un client identifié par son id
30.     public Medecin getMedecinById(long id);
31.
32.     // trouver un Rv identifié par son id
33.     public Rv getRvById(long id);
34.
35.     // trouver un créneau horaire identifié par son id
36.     public Creneau getCreneauById(long id);
37.
38.     // ajouter un RV
39.     public Rv ajouterRv(Date jour, Creneau créneau, Client client);
40.
41.     // supprimer un RV
42.     public void supprimerRv(Rv rv);
43.
44.     // metier
45.     public AgendaMedecinJour getAgendaMedecinJour(long idMedecin, Date jour);
46.
47. }
```

Les commentaires expliquent le rôle de chacune des méthodes.

L'implémentation de l'interface [IMetier] est la classe [Metier] suivante :

```

1. package rdvmedecins.metier;
2.
3. import java.util.Date;
4. import java.util.Hashtable;
5. import java.util.List;
6. import java.util.Map;
7.
8. import org.springframework.beans.factory.annotation.Autowired;
9. import org.springframework.stereotype.Service;
10.
11. import rdvmedecins.domain.AgendaMedecinJour;
12. import rdvmedecins.domain.CreneauMedecinJour;
13. import rdvmedecins.entities.Client;
14. import rdvmedecins.entities.Creneau;
15. import rdvmedecins.entities.Medecin;
16. import rdvmedecins.entities.Rv;
17. import rdvmedecins.repositories.ClientRepository;
18. import rdvmedecins.repositories.CreneauRepository;
19. import rdvmedecins.repositories.MedecinRepository;
20. import rdvmedecins.repositories.RvRepository;
21.
22. import com.google.common.collect.Lists;
23.
24. @Service("métier")
25. public class Metier implements IMetier {
26.
27.     // répositories
28.     @Autowired
29.     private MedecinRepository medecinRepository;
30.     @Autowired
31.     private ClientRepository clientRepository;
32.     @Autowired
33.     private CreneauRepository creneauRepository;
34.     @Autowired
35.     private RvRepository rvRepository;
36.
37.     // implémentation interface
38.     @Override
39.     public List<Client> getAllClients() {
40.         return Lists.newArrayList(clientRepository.findAll());
41.     }
42.
43.     @Override
44.     public List<Medecin> getAllMedecins() {
45.         return Lists.newArrayList(medecinRepository.findAll());
```

```

46.     }
47.
48.     @Override
49.     public List<Creneau> getAllCreneaux(long idMedecin) {
50.         return Lists.newArrayList(creneauRepository.getAllCreneaux(idMedecin));
51.     }
52.
53.     @Override
54.     public List<Rv> getRvMedecinJour(long idMedecin, Date jour) {
55.         return Lists.newArrayList(rvRepository.getRvMedecinJour(idMedecin, jour));
56.     }
57.
58.     @Override
59.     public Client getClientById(long id) {
60.         return clientRepository.findOne(id);
61.     }
62.
63.     @Override
64.     public Medecin getMedecinById(long id) {
65.         return medecinRepository.findOne(id);
66.     }
67.
68.     @Override
69.     public Rv getRvById(long id) {
70.         return rvRepository.findOne(id);
71.     }
72.
73.     @Override
74.     public Creneau getCreneauById(long id) {
75.         return creneauRepository.findOne(id);
76.     }
77.
78.     @Override
79.     public Rv ajouterRv(Date jour, Creneau creneau, Client client) {
80.         return rvRepository.save(new Rv(jour, client, creneau));
81.     }
82.
83.     @Override
84.     public void supprimerRv(Rv rv) {
85.         rvRepository.delete(rv.getId());
86.     }
87.
88.     public AgendaMedecinJour getAgendaMedecinJour(long idMedecin, Date jour) {
89.         ...
90.     }
91.
92. }

```

- ligne 24 : l'annotation `[@Service]` est une annotation Spring qui fait de la classe annotée un composant géré par Spring. On peut ou non donner un nom à un composant. Celui-ci est nommé `[métier]` ;
- ligne 25 : la classe `[Metier]` implémente l'interface `[IMetier]` ;
- ligne 28 : l'annotation `[@Autowired]` est une annotation Spring. La valeur du champ ainsi annoté sera initialisée (injectée) par Spring avec la référence d'un composant Spring du type ou du nom précisés. Ici l'annotation `[@Autowired]` ne précise pas de nom. Ce sera donc une injection par type qui sera faite ;
- ligne 29 : le champ `[medecinRepository]` sera initialisé avec la référence d'un composant Spring de type `[MedecinRepository]`. Ce sera la référence de la classe générée par Spring Data pour implémenter l'interface `[MedecinRepository]` que nous avons déjà présentée ;
- lignes 30-35 : ce processus est répété pour les trois autres interfaces étudiées ;
- lignes 39-41 : implémentation de la méthode `[getAllClients]` ;
- ligne 40 : nous utilisons la méthode `[findAll]` de l'interface `[ClientRepository]`. Cette méthode rend un type `[Iterable<Client>]` que nous transformons en `[List<Client>]` avec la méthode statique `[Lists.newArrayList]`. La classe `[Lists]` est définie dans la bibliothèque Google Guava. Dans `[pom.xml]` cette dépendance a été importée :

```

<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>16.0.1</version>
</dependency>

```

- lignes 38-86 : les méthodes de l'interface `[IMetier]` sont implémentées avec l'aide des classes de la couche `[DAO]` ;

Seule la méthode de la ligne 88 est spécifique à la couche [métier]. Elle a été placée ici parce qu'elle fait un traitement métier qui n'est pas qu'un simple accès aux données. Sans cette méthode, il n'y avait pas de raison de créer une couche [métier]. La méthode [getAgendaMedecinJour] est la suivante :

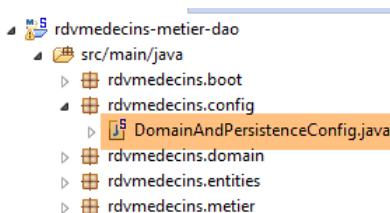
```

1. public AgendaMedecinJour getAgendaMedecinJour(long idMedecin, Date jour) {
2.     // liste des créneaux horaires du médecin
3.     List<Creneau> creneauxHoraires = getAllCreneaux(idMedecin);
4.     // liste des réservations de ce même médecin pour ce même jour
5.     List<Rv> reservations = getRvMedecinJour(idMedecin, jour);
6.     // on crée un dictionnaire à partir des Rv pris
7.     Map<Long, Rv> hReservations = new Hashtable<Long, Rv>();
8.     for (Rv resa : reservations) {
9.         hReservations.put(resa.getCreneau().getId(), resa);
10.    }
11.    // on crée l'agenda pour le jour demandé
12.    AgendaMedecinJour agenda = new AgendaMedecinJour();
13.    // le médecin
14.    agenda.setMedecin(getMedecinById(idMedecin));
15.    // le jour
16.    agenda.setJour(jour);
17.    // les créneaux de réservation
18.    CreneauMedecinJour[] creneauxMedecinJour = new CreneauMedecinJour[creneauxHoraires.size()];
19.    agenda.setCreneauxMedecinJour(creneauxMedecinJour);
20.    // remplissage des créneaux de réservation
21.    for (int i = 0; i < creneauxHoraires.size(); i++) {
22.        // ligne i agenda
23.        creneauxMedecinJour[i] = new CreneauMedecinJour();
24.        // créneau horaire
25.        Creneau creneau = creneauxHoraires.get(i);
26.        long idCreneau = creneau.getId();
27.        creneauxMedecinJour[i].setCreneau(creneau);
28.        // le créneau est-il libre ou réservé ?
29.        if (hReservations.containsKey(idCreneau)) {
30.            // le créneau est occupé - on note la résa
31.            Rv resa = hReservations.get(idCreneau);
32.            creneauxMedecinJour[i].setRv(resa);
33.        }
34.    }
35.    // on rend le résultat
36.    return agenda;
37. }
```

Le lecteur est invité à lire les commentaires. L'algorithme est le suivant :

- on récupère tous les créneaux horaires du médecin indiqué ;
- on récupère tous ses rendez-vous pour le jour indiqué ;
- avec ces deux informations, on est capable de dire si un créneau horaire est libre ou occupé ;

2.8 La configuration du projet



La classe [DomainAndPersistenceConfig] configure l'ensemble du projet :

```

1. package rdvmedecins.config;
2.
3. import javax.sql.DataSource;
4.
5. import org.apache.commons.dbcp.BasicDataSource;
6. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
```

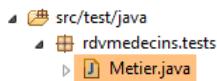
```

7. import org.springframework.boot.orm.jpa.EntityScan;
8. import org.springframework.context.annotation.Bean;
9. import org.springframework.context.annotation.ComponentScan;
10. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
11. import org.springframework.orm.jpa.JpaVendorAdapter;
12. import org.springframework.orm.jpa.vendor.Database;
13. import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
14. import org.springframework.transaction.annotation.EnableTransactionManagement;
15.
16. @EnableJpaRepositories(basePackages = { "rdvmedecins.repositories" })
17. @EnableAutoConfiguration
18. @ComponentScan(basePackages = { "rdvmedecins" })
19. @EntityScan(basePackages = { "rdvmedecins.entities" })
20. @EnableTransactionManagement
21. public class DomainAndPersistenceConfig {
22.
23.     // la source de données MySQL
24.     @Bean
25.     public DataSource dataSource() {
26.         BasicDataSource dataSource = new BasicDataSource();
27.         dataSource.setDriverClassName("com.mysql.jdbc.Driver");
28.         dataSource.setUrl("jdbc:mysql://localhost:3306/dbrdvmmedecins");
29.         dataSource.setUsername("root");
30.         dataSource.setPassword("");
31.         return dataSource;
32.     }
33.
34.     // le provider JPA - n'est pas nécessaire si on est satisfait des valeurs par défaut utilisées par
35.     // Spring boot
36.     // ici on le définit pour activer / désactiver les logs SQL
37.     @Bean
38.     public JpaVendorAdapter jpaVendorAdapter() {
39.         HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new HibernateJpaVendorAdapter();
40.         hibernateJpaVendorAdapter.setShowSql(false);
41.         hibernateJpaVendorAdapter.setGenerateDdl(false);
42.         hibernateJpaVendorAdapter.setDatabase(Database.MYSQL);
43.         return hibernateJpaVendorAdapter;
44.     }
45.     // l'EntityManagerFactory et le TransactionManager sont définis avec des valeurs par défaut par Spring
46.     // boot
47. }

```

- lignes 45 : nous n'allons pas définir les beans [EntityManagerFactory] et [TransactionManager]. Nous allons pour cela nous appuyer sur l'annotation [@EnableAutoConfiguration] de Spring Boot (ligne 17) ;
- lignes 24-32 : définissent la source de données MySQL5. C'est un bean qui en général ne peut être deviné par Spring Boot ;
- lignes 36-43 : nous configurons également l'implémentation JPA pour mettre l'attribut [showSql] d'Hibernate à faux (ligne 39). Par défaut, il est à vrai ;
- pour l'instant, les seuls composants gérés par Spring sont les beans des lignes 25 et 37 plus les beans [EntityManagerFactory] et [TransactionManager] par autoconfiguration. Il nous faut ajouter les beans des couches [métier] et [DAO] ;
- la ligne 16 ajoute au contexte de Spring les interfaces du package [rdvmedecins.repositories] qui héritent de l'interface [CrudRepository] ;
- la ligne 18 ajoute au contexte de Spring toutes les classes du package [rdvmedecins] et ses descendants ayant une annotation Spring. Dans le package [rdvmedecins.metier], la classe [Metier] avec son annotation [@Service] va être trouvée et ajoutée au contexte Spring ;
- ligne 45 : un bean [entityManagerFactory] va être défini par défaut par Spring Boot. On doit indiquer à ce bean où se trouvent les entités JPA qu'il doit gérer. C'est la ligne 19 qui fait cela ;
- ligne 20 : indique que les méthodes des interfaces héritant de l'interface [CrudRepository] doivent être exécutées au sein d'une transaction ;

2.9 Les tests de la couche [métier]



La classe [rdvmedecins.tests.Metier] est une classe de test Spring / JUnit 4 :

```
1. package rdvmedecins.tests;
2.
3. import java.text.ParseException;
4. import java.util.Date;
5. import java.util.List;
6.
7. import org.junit.Assert;
8. import org.junit.Test;
9. import org.junit.runner.RunWith;
10. import org.springframework.beans.factory.annotation.Autowired;
11. import org.springframework.boot.test.SpringApplicationConfiguration;
12. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
13.
14. import rdvmedecins.config.DomainAndPersistenceConfig;
15. import rdvmedecins.domain.AgendaMedecinJour;
16. import rdvmedecins.entities.Client;
17. import rdvmedecins.entities.Creneau;
18. import rdvmedecins.entities.Medecin;
19. import rdvmedecins.entities.Rv;
20. import rdvmedecins.metier.IMetier;
21.
22. @SpringApplicationConfiguration(classes = DomainAndPersistenceConfig.class)
23. @RunWith(SpringJUnit4ClassRunner.class)
24. public class Metier {
25.
26.     @Autowired
27.     private IMetier métier;
28.
29.     @Test
30.     public void test1(){
31.         // affichage clients
32.         List<Client> clients = métier.getAllClients();
33.         display("Liste des clients :", clients);
34.         // affichage médecins
35.         List<Medecin> medecins = métier.getAllMedecins();
36.         display("Liste des médecins :", medecins);
37.         // affichage créneaux d'un médecin
38.         Medecin médecin = medecins.get(0);
39.         List<Creneau> créneaux = métier.getAllCrenaeux(médecin.getId());
40.         display(String.format("Liste des créneaux du médecin %s", médecin), créneaux);
41.         // liste des Rv d'un médecin, un jour donné
42.         Date jour = new Date();
43.         display(String.format("Liste des rv du médecin %s, le [%s]", médecin, jour),
44.               métier.getRvMedecinJour(médecin.getId(), jour));
45.         // ajouter un RV
46.         Rv rv = null;
47.         Creneau créneau = créneaux.get(2);
48.         Client client = clients.get(0);
49.         System.out.println(String.format("Ajout d'un Rv le [%s] dans le créneau %s pour le client %s", jour,
50.                                         créneau,
51.                                         client));
52.         rv = métier.ajouterRv(jour, créneau, client);
53.         // vérification
54.         Rv rv2 = métier.getRvById(rv.getId());
55.         Assert.assertEquals(rv, rv2);
56.         display(String.format("Liste des Rv du médecin %s, le [%s]", médecin, jour),
57.               métier.getRvMedecinJour(médecin.getId(), jour));
58.         // ajouter un RV dans le même créneau du même jour
59.         Boolean erreur = false;
60.         try {
61.             rv = métier.ajouterRv(jour, créneau, client);
62.             System.out.println("Rv ajouté");
```

```

63.     } catch (Exception ex) {
64.         Throwable th = ex;
65.         while (th != null) {
66.             System.out.println(ex.getMessage());
67.             th = th.getCause();
68.         }
69.         // on note l'erreur
70.         erreur = true;
71.     }
72.     // on vérifie qu'il y a eu une erreur
73.     Assert.assertTrue(erreur);
74.     // liste des RV
75.     display(String.format("Liste des Rv du médecin %s, le [%s]", médecin, jour),
    métier.getRvMedecinJour(médecin.getId(), jour));
76.     // affichage agenda
77.     AgendaMedecinJour agenda = métier.getAgendaMedecinJour(médecin.getId(), jour);
78.     System.out.println(agenda);
79.     Assert.assertEquals(rv, agenda.getCreneauxMedecinJour()[2].getRv());
80.     // supprimer un RV
81.     System.out.println("Suppression du Rv ajouté");
82.     métier.supprimerRv(rv);
83.     // vérification
84.     rv2 = métier.getRvById(rv.getId());
85.     Assert.assertNull(rv2);
86.     display(String.format("Liste des Rv du médecin %s, le [%s]", médecin, jour),
    métier.getRvMedecinJour(médecin.getId(), jour));
87. }
88.
89. // méthode utilitaire - affiche les éléments d'une collection
90. private void display(String message, Iterable<?> elements) {
91.     System.out.println(message);
92.     for (Object element : elements) {
93.         System.out.println(element);
94.     }
95. }
96.
97. }

```

- ligne 22 : l'annotation `[@SpringApplicationConfiguration]` permet d'exploiter le fichier de configuration `[DomainAndPersistenceConfig]` étudié précédemment. La classe de test bénéficie ainsi de tous les beans définis par ce fichier ;
- ligne 23 : l'annotation `[@RunWith]` permet l'intégration de Spring avec JUnit : la classe va pouvoir être exécutée comme un test JUnit. `[@RunWith]` est une annotation JUnit (ligne 9) alors que la classe `[SpringJUnit4ClassRunner]` est une classe Spring (ligne 12) ;
- lignes 26-27 : injection dans la classe de test d'une référence sur la couche `[métier]` ;
- beaucoup de tests ne sont que de simples tests visuels :
 - lignes 32-33 : liste des clients ;
 - lignes 35-36 : liste des médecins ;
 - lignes 39-40 : liste des créneaux d'un médecin ;
 - ligne 43 : liste des rendez-vous d'un médecin ;
- ligne 50 : ajout d'un nouveau rendez-vous. La méthode `[ajouterRv]` rend le rendez-vous avec une information supplémentaire, sa clé primaire `id` ;
- ligne 53 : on utilise cette clé primaire pour rechercher le rendez-vous en base ;
- ligne 54 : on vérifie que le rendez-vous cherché et le rendez-vous trouvé sont les mêmes. On rappelle que la méthode `[equals]` de l'entité `[Rv]` a été redéfinie : deux rendez-vous sont égaux s'ils ont le même `id`. Ici, cela nous montre que le rendez-vous ajouté a bien été mis en base ;
- lignes 61-73 : on essaie d'ajouter une deuxième fois le même rendez-vous. Cela doit être rejeté par le SGBD car on a une contrainte d'unicité :

```

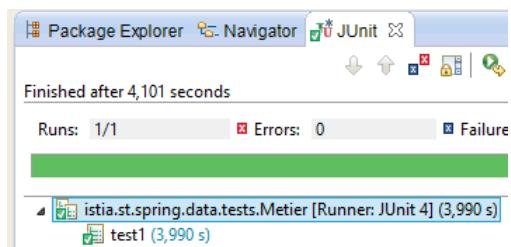
1. CREATE TABLE IF NOT EXISTS `rv` (
2.     `ID` bigint(20) NOT NULL AUTO_INCREMENT,
3.     `JOUR` date NOT NULL,
4.     `ID_CLIENT` bigint(20) NOT NULL,
5.     `ID_CRENEAU` bigint(20) NOT NULL,
6.     `VERSION` int(11) NOT NULL DEFAULT '0',
7.     PRIMARY KEY (`ID`),
8.     UNIQUE KEY `UNQ1_RV` (`JOUR`, `ID_CRENEAU`),
9.     KEY `FK_RV_ID_CRENEAU` (`ID_CRENEAU`),
10.    KEY `FK_RV_ID_CLIENT` (`ID_CLIENT`),
11.) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci AUTO_INCREMENT=60 ;

```

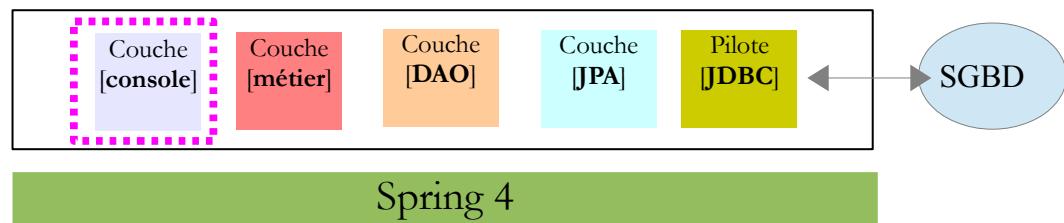
La ligne 8 ci-dessus indique que la combinaison [JOUR, ID_CRENEAU] doit être unique, ce qui empêche de mettre deux rendez-vous le même jour dans le même créneau horaire.

- ligne 73 : on vérifie qu'une exception s'est bien produite ;
- ligne 77 : on demande l'agenda du médecin pour lequel on vient d'ajouter un rendez-vous ;
- ligne 79 : on vérifie que le rendez-vous ajouté est bien présent dans son agenda ;
- ligne 82 : on supprime le rendez-vous ajouté ;
- ligne 84 : on va chercher en base le rendez-vous supprimé ;
- ligne 85 : on vérifie qu'on a récupéré un pointeur *null*, montrant par là que le rendez-vous cherché n'existe pas ;

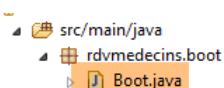
L'exécution du test réussit :



2.10 Le programme console



Spring 4



Le programme console est basique. Il illustre comment récupérer une clé étrangère :

```
1. package rdvmedecins.boot;
2.
3. import java.text.SimpleDateFormat;
4. import java.util.Date;
5.
6. import org.springframework.boot.SpringApplication;
7. import org.springframework.context.ConfigurableApplicationContext;
8.
9. import rdvmedecins.config.DomainAndPersistenceConfig;
10. import rdvmedecins.entities.Client;
11. import rdvmedecins.entities.Creneau;
12. import rdvmedecins.entities.Rv;
13. import rdvmedecins.metier.IMetier;
14.
15. public class Boot {
16.     // le boot
17.     public static void main(String[] args) {
18.         // on prépare la configuration
19.         SpringApplication app = new SpringApplication(DomainAndPersistenceConfig.class);
```

```

20.     app.setLogStartupInfo(false);
21.     // on la lance
22.     ConfigurableApplicationContext context = app.run(args);
23.     // métier
24.     IMetier métier = context.getBean(IMetier.class);
25.     try {
26.         // ajouter un RV
27.         Date jour = new Date();
28.         System.out.println(String.format("Ajout d'un Rv le [%s] dans le créneau 1 pour le client 1", new
SimpleDateFormat("dd/MM/yyyy").format(jour)));
29.         Client client = (Client) new Client().build(1L, 1L);
30.         Creneau créneau = (Creneau) new Creneau().build(1L, 1L);
31.         Rv rv = métier.ajouterRv(jour, créneau, client);
32.         System.out.println(String.format("Rv ajouté = %s", rv));
33.         // vérification
34.         créneau = métier.getCreneauById(1L);
35.         long idMedecin = créneau.getIdMedecin();
36.         display("Liste des rendez-vous", métier.getRvMedecinJour(idMedecin, jour));
37.     } catch (Exception ex) {
38.         System.out.println("Exception : " + ex.getCause());
39.     }
40.     // fermeture du contexte Spring
41.     context.close();
42. }
43.
44. // méthode utilitaire - affiche les éléments d'une collection
45. private static <T> void display(String message, Iterable<T> elements) {
46.     System.out.println(message);
47.     for (T element : elements) {
48.         System.out.println(element);
49.     }
50. }
51.
52. }

```

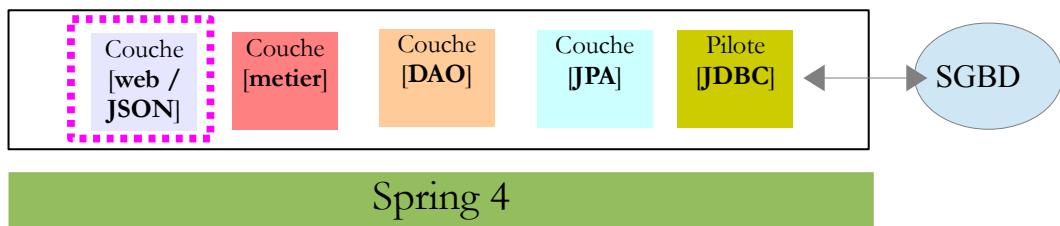
Le programme ajoute un rendez-vous et ensuite vérifie qu'il a été ajouté.

- ligne 19 : la classe [SpringApplication] va exploiter la classe de configuration [DomainAndPersistenceConfig] ;
- ligne 20 : suppression des logs de démarrage de l'application ;
- ligne 22 : la classe [SpringApplication] est exécutée. Elle rend un contexte Spring, c-à-d la liste des beans enregistrés ;
- ligne 24 : on récupère une référence sur le bean implémentant l'interface [IMetier]. Il s'agit donc d'une référence sur la couche [métier] ;
- lignes 27-31 : ajout d'un nouveau rendez-vous pour aujourd'hui, pour le client n°1 dans le créneau n° 1. Le client et le créneau ont été créés de toute pièce pour montrer que seuls les identifiants sont utilisés. On a initialisé ici la version mais on n'aurait pu mettre n'importe quoi. Elle n'est pas utilisée ici ;
- ligne 34 : on veut connaître le médecin ayant le créneau n° 1. Pour cela on a besoin d'aller en base chercher le créneau n° 1. Parce qu'on est en mode [FetchType.LAZY], le médecin n'est pas ramené avec le créneau. Cependant, on a pris soin de prévoir un champ [idMedecin] dans l'entité [Creneau] pour récupérer la clé primaire du médecin ;
- ligne 35 : on récupère la primaire du médecin ;
- ligne 36 : on affiche la liste des rendez-vous du médecin ;

Les résultats console sont les suivants :

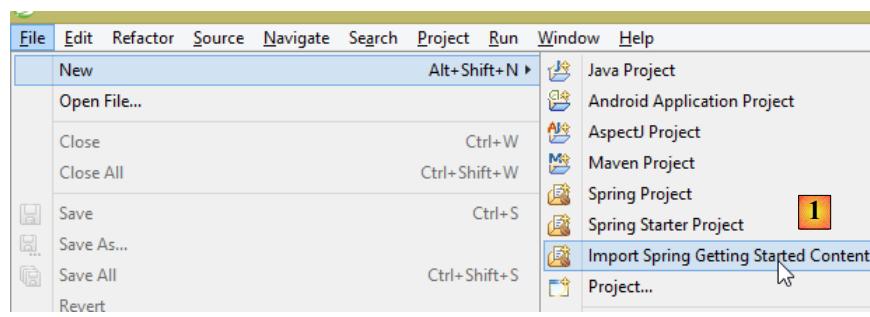
1. Ajout d'un Rv le [10/06/2014] dans le créneau 1 pour le client 1
2. Rv ajouté = Rv[113, Tue Jun 10 16:51:01 CEST 2014, 1, 1]
3. Liste des rendez-vous
4. Rv[113, 2014-06-10, 1, 1]

2.11 Introduction à Spring MVC

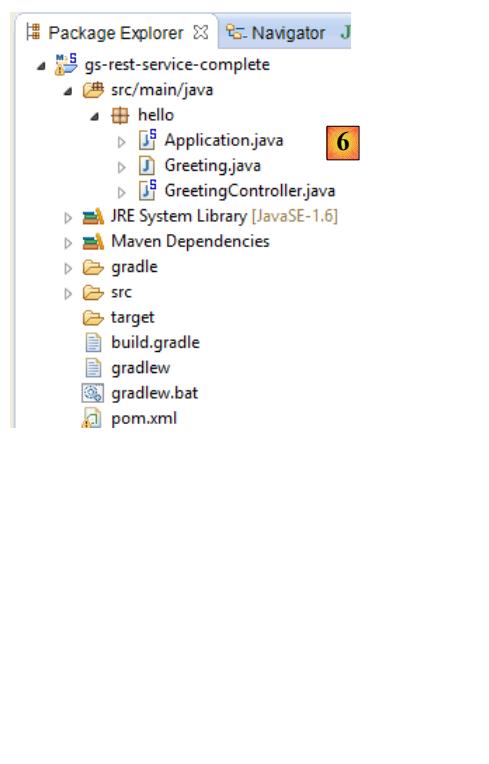
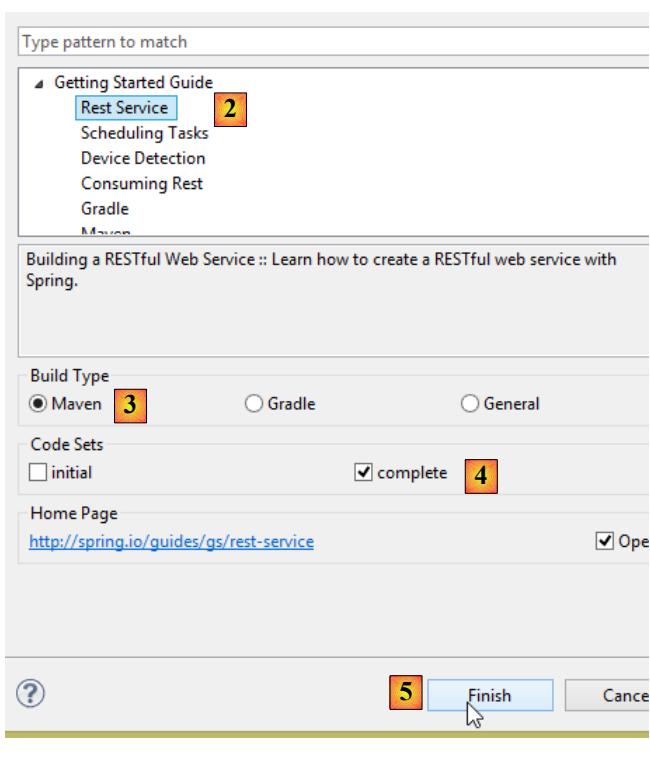


Nous abordons maintenant la construction de la couche web. Celle-ci est principalement constituée de méthodes qui traitent des URL précises et répondent avec une ligne de texte au format JSON (Javascript Object Notation). Cette couche web est une interface web qu'on appelle parfois une API web. Nous allons implémenter cette interface avec Spring MVC, une autre branche de l'écosystème Spring. Nous commençons par étudier l'un des guides disponibles sur [<http://spring.io>].

2.11.1 Le projet de démonstration



- en [1], nous importons l'un des guides Spring ;



- en [2], nous choisissons l'exemple [Rest Service] ;
- en [3], on choisit le projet Maven ;
- en [4], on prend la version finale du guide ;
- en [5], on valide ;
- en [6], le projet importé ;

Les services web accessibles via des URL standard et qui délivrent du texte JSON sont souvent appelés des services REST (REpresentational State Transfer). Dans ce document, je me contenterai d'appeler le service que nous allons construire, un service web / JSON. Un service est dit Restful s'il respecte certaines règles. Je n'ai pas cherché à respecter celles-ci.

Examinons maintenant le projet importé, d'abord sa configuration Maven.

2.11.2 Configuration Maven

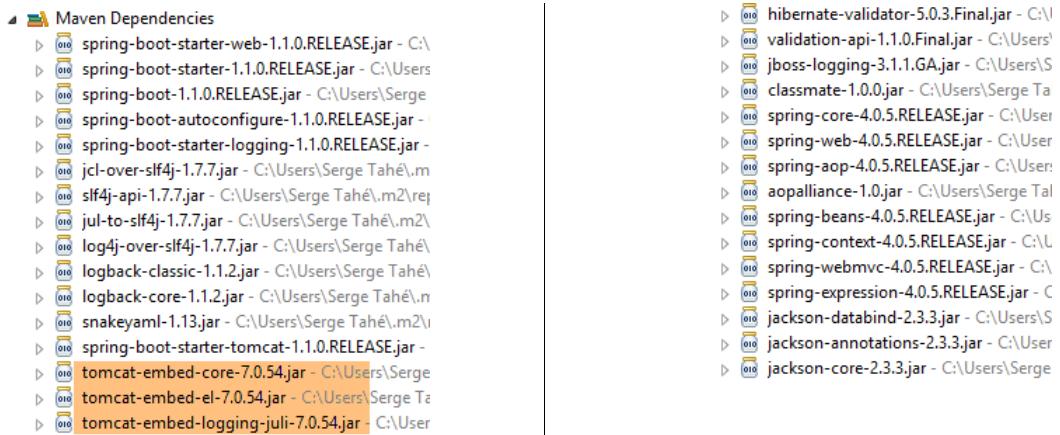
Le fichier [pom.xml] est le suivant :

```

1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4.    <modelVersion>4.0.0</modelVersion>
5.
6.    <groupId>org.springframework</groupId>
7.    <artifactId>gs-rest-service</artifactId>
8.    <version>0.1.0</version>
9.
10.   <parent>
11.     <groupId>org.springframework.boot</groupId>
12.     <artifactId>spring-boot-starter-parent</artifactId>
13.     <version>1.1.0.RELEASE</version>
14.   </parent>
15.
16.   <dependencies>
17.     <dependency>
18.       <groupId>org.springframework.boot</groupId>
19.       <artifactId>spring-boot-starter-web</artifactId>
20.     </dependency>
21.     <dependency>
22.       <groupId>com.fasterxml.jackson.core</groupId>
23.       <artifactId>jackson-databind</artifactId>
24.     </dependency>
25.   </dependencies>
26.
27.   <properties>
28.     <start-class>hello.Application</start-class>
29.   </properties>
30.
31.   <build>
32.     <plugins>
33.       <plugin>
34.         <artifactId>maven-compiler-plugin</artifactId>
35.       </plugin>
36.       <plugin>
37.         <groupId>org.springframework.boot</groupId>
38.         <artifactId>spring-boot-maven-plugin</artifactId>
39.       </plugin>
40.     </plugins>
41.   </build>
42.
43.   <repositories>
44.     <repository>
45.       <id>spring-releases</id>
46.       <url>http://repo.spring.io/release</url>
47.     </repository>
48.   </repositories>
49.   <pluginRepositories>
50.     <pluginRepository>
51.       <id>spring-releases</id>
52.       <url>http://repo.spring.io/release</url>
53.     </pluginRepository>
54.   </pluginRepositories>
55. </project>
```

- lignes 10-14 : comme dans le projet [Spring Data], on trouve le projet parent [Spring Boot] ;
- lignes 17-20 : l'artifact [spring-boot-starter-web] amène avec lui les bibliothèques nécessaires à un projet spring MVC. Il amène en particulier avec lui un serveur Tomcat embarqué. C'est sur ce serveur que l'application sera exécutée ;
- lignes 21-24 : la bibliothèque Jackson gère le JSON : transformation d'un objet Java en chaîne JSON et inversement ;

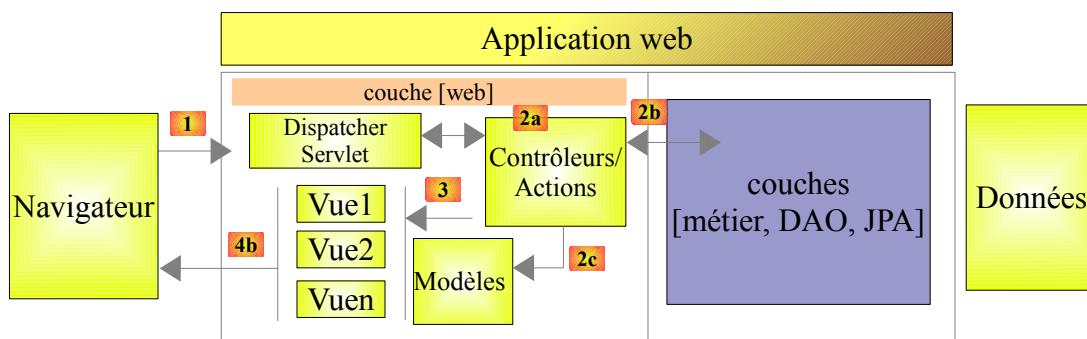
Les bibliothèques amenées par cette configuration sont très nombreuses :



Ci-dessus on voit les trois archives du serveur Tomcat.

2.11.3 L'architecture d'un service Spring REST

Spring MVC implémente le modèle d'architecture dit MVC (Modèle – Vue – Contrôleur) de la façon suivante :

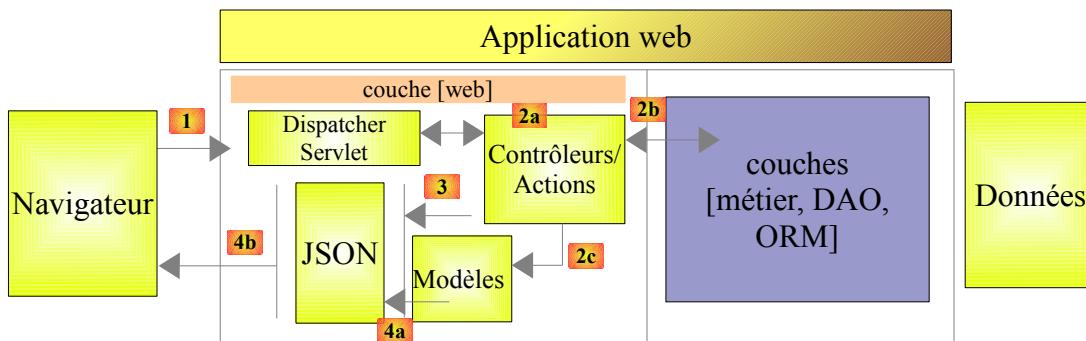


Le traitement d'une demande d'un client se déroule de la façon suivante :

1. **demande** - les URL demandées sont de la forme `http://machine:port/contexte/Action/param1/param2/...?p1=v1&p2=v2...`. La [Dispatcher Servlet] est la classe de Spring qui traite les URL entrantes. Elle "route" l'URL vers l'action qui doit la traiter. Ces actions sont des méthodes de classes particulières appelées [Contrôleurs]. Le **C** de MVC est ici la chaîne [Dispatcher Servlet, Contrôleur, Action]. Si aucune action n'a été configurée pour traiter l'URL entrante, la servlet [Dispatcher Servlet] répondra que l'URL demandée n'a pas été trouvée (erreur 404 NOT FOUND) ;
2. **traitement**
 - l'action choisie peut exploiter les paramètres *parami* que la servlet [Dispatcher Servlet] lui a transmis. Ceux-ci peuvent provenir de plusieurs sources :
 - du chemin `[/param1/param2/...]` de l'URL,
 - des paramètres `[p1=v1&p2=v2]` de l'URL,
 - de paramètres postés par le navigateur avec sa demande ;
 - dans le traitement de la demande de l'utilisateur, l'action peut avoir besoin de la couche [metier] [2b]. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
 - une page d'erreur si la demande n'a pu être traitée correctement
 - une page de confirmation sinon
 - l'action demande à une certaine vue de s'afficher [3]. Cette vue va afficher des données qu'on appelle le **modèle de la vue**. C'est le **M** de MVC. L'action va créer ce modèle M [2c] et demander à une vue V de s'afficher [3] ;

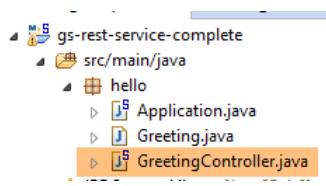
3. **réponse** - la vue **V** choisie utilise le modèle **M** construit par l'action pour initialiser les parties dynamiques de la réponse HTML qu'elle doit envoyer au client puis envoie cette réponse.

Pour un service web / JSON, l'architecture précédente est légèrement modifiée :



- en [4a], le modèle qui est une classe Java est transformé en chaîne JSON par une bibliothèque JSON ;
- en [4b], cette chaîne JSON est envoyée au navigateur ;

2.11.4 Le contrôleur C



L'application importée a le contrôleur suivant :

```

1. package hello;
2.
3. import java.util.concurrent.atomic.AtomicLong;
4. import org.springframework.stereotype.Controller;
5. import org.springframework.web.bind.annotation.RequestMapping;
6. import org.springframework.web.bind.annotation.RequestParam;
7. import org.springframework.web.bind.annotation.ResponseBody;
8.
9. @Controller
10. public class GreetingController {
11.
12.     private static final String template = "Hello, %s!";
13.     private final AtomicLong counter = new AtomicLong();
14.
15.     @RequestMapping("/greeting")
16.     public @ResponseBody
17.     Greeting greeting(@RequestParam(value = "name", required = false, defaultValue = "World") String name)
18.     {
19.         return new Greeting(counter.incrementAndGet(), String.format(template, name));
20.     }

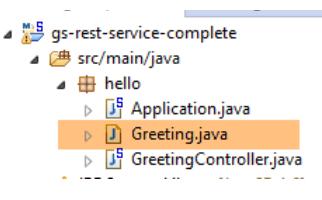
```

- ligne 9 : l'annotation `@Controller` fait de la classe `GreetingController` un contrôleur Spring, c'est-à-dire que ses méthodes sont enregistrées pour traiter des URL ;
- ligne 15 : l'annotation `@RequestMapping` indique l'URL que traite la méthode, ici l'URL `/greeting`. Nous verrons ultérieurement que cette URL peut être paramétrée et qu'il est possible de récupérer ces paramètres ;
- ligne 16 : l'annotation `@ResponseBody` indique que la méthode ne produit pas un modèle pour une vue (JSP, JSF, Thymeleaf, ...) qui sera envoyée ensuite au navigateur client mais produit elle-même la réponse faite au navigateur. Ici, elle produit un objet de type `Greeting` (ligne 18). De façon non apparente ici, cet objet va d'abord être transformé en JSON avant d'être envoyé au navigateur. C'est la présence d'une bibliothèque JSON dans les dépendances du projet qui fait que Spring Boot va, par autoconfiguration, configurer le projet de cette façon ;

- ligne 17 : la méthode [greeting] a un paramètre [String name]. L'annotation [@RequestParam(value = "name", required = false, defaultValue = "World")] indique que ce paramètre doit être initialisé avec un paramètre nommé [name] (@RequestParam(value = "name")). Celui-ci peut être le paramètre d'un GET ou d'un POST. Ce paramètre n'est pas obligatoire (required = false). Dans ce dernier cas, le paramètre [name] de la méthode sera initialisé avec la valeur [World] (defaultValue = "World").

2.11.5 Le modèle M

Le modèle M produit par la méthode précédente est l'objet [Greeting] suivant :



```

1. package hello;
2.
3. public class Greeting {
4.
5.     private final long id;
6.     private final String content;
7.
8.     public Greeting(long id, String content) {
9.         this.id = id;
10.        this.content = content;
11.    }
12.
13.    public long getId() {
14.        return id;
15.    }
16.
17.    public String getContent() {
18.        return content;
19.    }
20. }
```

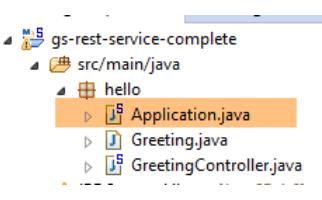
La transformation JSON de cet objet créera la chaîne de caractères {"id":n,"content":"texte"}. Au final, la chaîne JSON produite par la méthode du contrôleur sera de la forme :

{"id":2,"content":"Hello, World!"}

ou

{"id":2,"content":"Hello, John!"}

2.11.6 Configuration du projet



Le projet est configuré par la classe [Application] suivante :

```

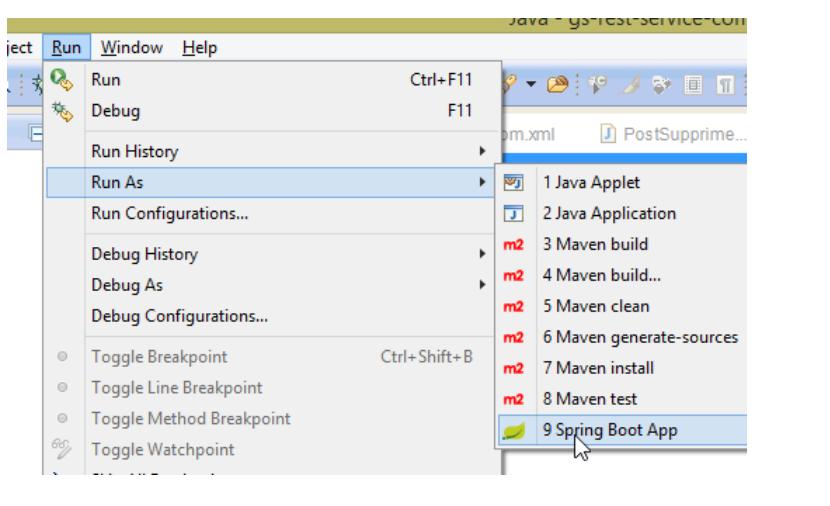
1. package hello;
2.
3. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
4. import org.springframework.boot.SpringApplication;
```

```
5. import org.springframework.context.annotation.ComponentScan;
6.
7. @ComponentScan
8. @EnableAutoConfiguration
9. public class Application {
10.
11.     public static void main(String[] args) {
12.         SpringApplication.run(Application.class, args);
13.     }
14. }
```

- ligne 11 : curieusement cette classe est exécutable avec une méthode [main] propre aux applications console. C'est bien le cas. La classe [SpringApplication] de la ligne 12 va lancer le serveur Tomcat présent dans les dépendances et déployer le service REST dessus ;
 - ligne 4 : on voit que la classe [SpringApplication] appartient au projet [Spring Boot] ;
 - ligne 12 : le premier paramètre est la classe qui configure le projet, le second d'éventuels paramètres ;
 - ligne 8 : l'annotation [@EnableAutoConfiguration] demande à Spring Boot de faire la configuration du projet ;
 - ligne 7 : l'annotation [@ComponentScan] fait que le dossier qui contient la classe [Application] va être exploré pour rechercher les composants Spring. Un sera trouvé, la classe [GreetingController] qui a l'annotation [@Controller] qui en fait un composant Spring ;

2.11.7 Exécution du projet

Exécutons le projet :



On obtient les logs console suivants :

```
1.  /\_\ / ____\_\_/\_\_/\_\_/\_\_/\_\_/\_\_/\_\_/\_\_
2.  (\ )\_\_|\_\_.|_\_|\_\_.|_\_|\_\_/\_\_|\_\_/\_\_/\_\_
3.  \\\_/\_\_)|_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_)\_\_
4.  ' |_____|_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_
5.  ======|_\_|=====|_\_|=/_\_|/_/_
6.  :: Spring Boot ::      (v1.1.0.RELEASE)
7.
8.  2014-06-11 14:31:36.435  INFO 11744 --- [           main] hello.Application      :
   Starting Application on Gportpers3 with PID 11744 (D:\Temp\wksSTS\gs-rest-service-complete\target\classes
   started by ST in D:\Temp\wksSTS\gs-rest-service-complete)
9.  2014-06-11 14:31:36.473  INFO 11744 --- [           main]  ationConfigEmbeddedWebApplicationContext :
   Refreshing
   org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@7684af0b: startup
   date [Wed Jun 11 14:31:36 CEST 2014]; root of context hierarchy
10. 2014-06-11 14:31:36.966  INFO 11744 --- [          main] o.s.b.f.s.DefaultListableBeanFactory    :
   Overriding bean definition for bean 'beanNameViewResolver': replacing [Root bean: class [null]; scope=;
   abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; autowireCandidate=true; primary=false;
   factoryBeanName=org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$WhitelabelErrorViewCo
   nfiguration; factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred);
   defined in class path resource
   [org/springframework/boot/autoconfigure/web/ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguratio
   n.class]] with [Root bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3;
```

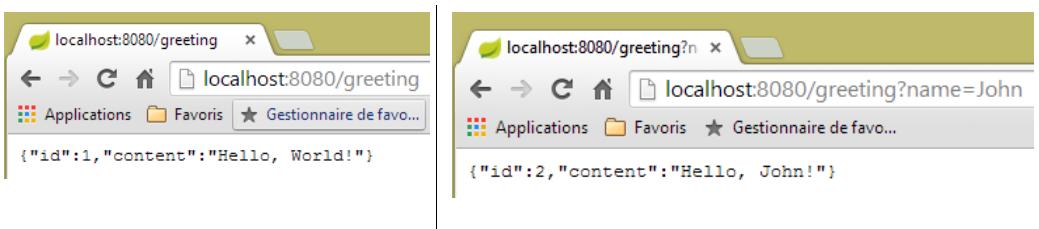
```

dependencyCheck=0; autowireCandidate=true; primary=false;
factoryBeanName=org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$WebMvcAutoConfiguration
Adapter; factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred);
defined in class path resource
[org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter.class]
11. 2014-06-11 14:31:37.760  INFO 11744 --- [           main] t.TomcatEmbeddedServletContainerFactory :
Server initialized with port: 8080
12. 2014-06-11 14:31:37.955  INFO 11744 --- [           main] o.apache.catalina.core.StandardService  :
Starting service Tomcat
13. 2014-06-11 14:31:37.956  INFO 11744 --- [           main] org.apache.catalina.core.StandardEngine  :
Starting Servlet Engine: Apache Tomcat/7.0.54
14. 2014-06-11 14:31:38.053  INFO 11744 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]      :
Initializing Spring embedded WebApplicationContext
15. 2014-06-11 14:31:38.054  INFO 11744 --- [ost-startStop-1] o.s.web.context.ContextLoader          : Root
WebApplicationContext: initialization completed in 1584 ms
16. 2014-06-11 14:31:38.596  INFO 11744 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean        :
Mapping servlet: 'dispatcherServlet' to [/]
17. 2014-06-11 14:31:38.598  INFO 11744 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean   :
Mapping filter: 'hiddenHttpMethodFilter' to: [//*]
18. 2014-06-11 14:31:38.919  INFO 11744 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  :
Mapped URL path [/**/favicon.ico] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
19. 2014-06-11 14:31:39.125  INFO 11744 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
Mapped "[[/greeting],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]]" onto public
hello.Greeting hello.GreetingController.greeting(java.lang.String)
20. 2014-06-11 14:31:39.129  INFO 11744 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
Mapped "[[/error],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]]" onto public
org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletReques
t)
21. 2014-06-11 14:31:39.130  INFO 11744 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
Mapped "[[/error],methods=[],params=[],headers=[],consumes=[],produces=[text/html],custom=[]]" onto public
org.springframework.web.servlet.ModelAndView
org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRe
quest)
22. 2014-06-11 14:31:39.160  INFO 11744 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  :
Mapped URL path [/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
23. 2014-06-11 14:31:39.160  INFO 11744 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  :
Mapped URL path [/webjars/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
24. 2014-06-11 14:31:39.448  INFO 11744 --- [           main] o.s.j.e.a.AnnotationMBeanExporter          :
Registering beans for JMX exposure on startup
25. 2014-06-11 14:31:39.490  INFO 11744 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer  :
Tomcat started on port(s): 8080/http
26. 2014-06-11 14:31:39.492  INFO 11744 --- [           main] hello.Application                  :
Started Application in 3.45 seconds (JVM running for 3.93)

```

- ligne 12 : le serveur Tomcat démarre sur le port 8080 (ligne 11) ;
- ligne 16 : la servlet [DispatcherServlet] est présente ;
- ligne 19 : la méthode [GreetingController.greeting] a été découverte ;

Pour tester l'application web, on demande l'URL [http://localhost:8080/greeting] :



On reçoit bien la chaîne JSON attendue. Il peut être intéressant de voir les entêtes HTTP envoyés par le serveur. Pour cela, on va utiliser le plugin de Chrome appelé [Advanced Rest Client] (cf Annexes) :

The screenshot shows two separate Postman requests side-by-side.

Left Request (GET):

- URL: `http://localhost:8080/greeting` (highlighted with box 1)
- Method: GET (highlighted with box 2)
- Raw tab selected
- Status: 200 OK (highlighted with box 3)
- Request headers:
 - User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36
 - Content-Type: text/plain; charset=utf-8
 - Accept: */*
 - Accept-Encoding: gzip,deflate,sdch
 - Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
- Response headers:
 - Server: Apache-Coyote/1.1
 - Content-Type: application/json;charset=UTF-8 (highlighted with box 4)
 - Transfer-Encoding: chunked
 - Date: Wed, 11 Jun 2014 12:44:20 GMT
- Raw tab selected
- Response JSON:


```
{"id":4,"content":"Hello, World!"}
```

Right Request (POST):

- URL: `http://localhost:8080/greeting`
- Method: POST (highlighted with box 5)
- Raw tab selected
- Request Headers:
 - name=Mark (highlighted with box 6)
- Content-Type dropdown set to "application/x-www-form-urlencoded" (highlighted with box 7)
- Status: 200 OK (highlighted with box 8)
- Request headers:
 - User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36
 - Origin: chrome-extension://hgmllofdffdnphfgcellkdfbfbjel
 - Content-Type: application/x-www-form-urlencoded (highlighted with box 9)
 - Accept: */*
 - Accept-Encoding: gzip,deflate,sdch
 - Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
- Response headers:
 - Server: Apache-Coyote/1.1
 - Content-Type: application/json;charset=UTF-8
 - Transfer-Encoding: chunked
 - Date: Wed, 11 Jun 2014 12:47:38 GMT
- Raw tab selected
- Response JSON:


```
{
  "id": 6,
  "content": "Hello, Mark!"
}
```

- en [1], l'URL demandée ;
- en [2], la méthode GET est utilisée ;
- en [3], la réponse JSON ;
- en [4], le serveur a indiqué qu'il envoyait une réponse au format JSON ;
- en [5], on demande la même URL mais cette fois-ci avec un POST ;
- en [7], les informations sont envoyées au serveur sous la forme [urlencoded] ;
- en [6], le paramètre **name** avec sa valeur ;
- en [8], le navigateur indique au serveur qu'il lui envoie des informations [urlencoded] ;
- en [9], la réponse JSON du serveur ;

2.11.8 Création d'une archive exécutable

Il est possible de créer une archive exécutable en-dehors d'Eclipse. La configuration nécessaire est dans le fichier [pom.xml] :

```

1.   <properties>
2.     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
3.     <start-class>istia.st.Application</start-class>
4.     <java.version>1.7</java.version>
5.   </properties>
6.
7.   <build>
8.     <plugins>
```

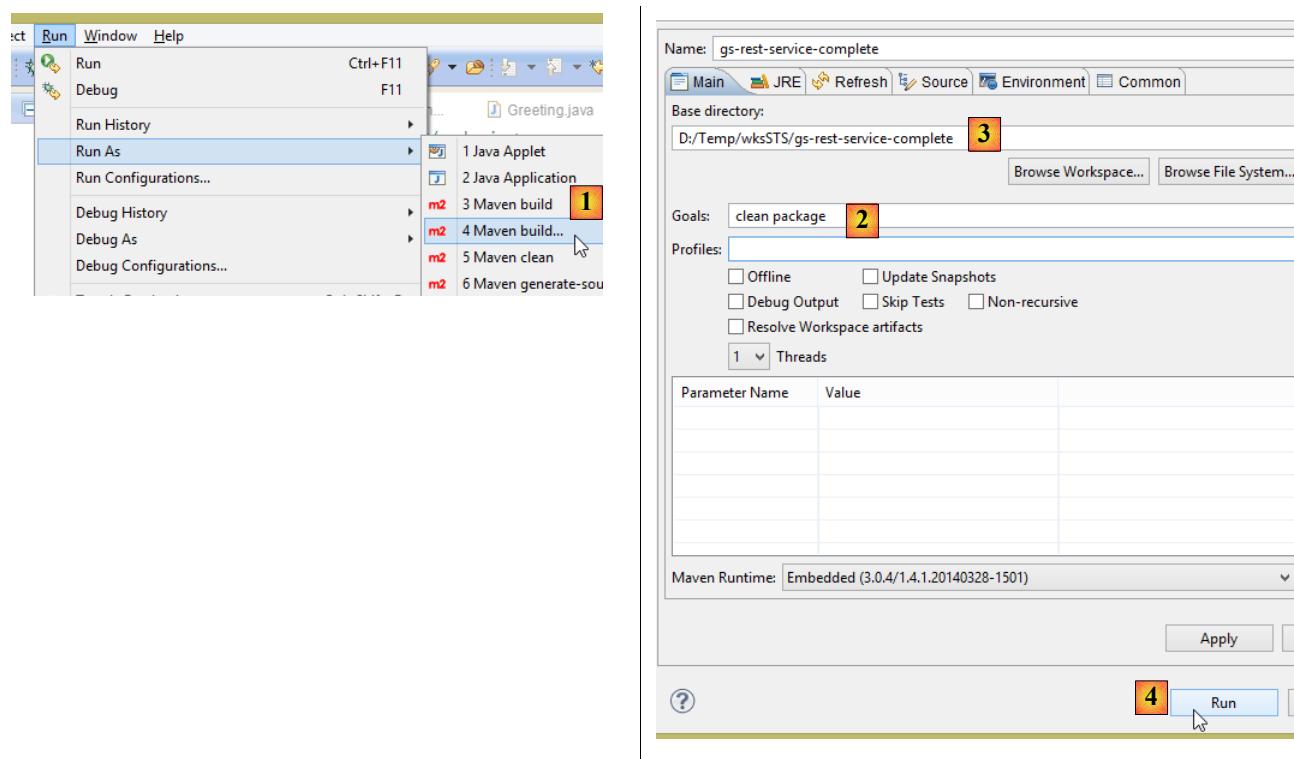
```

9.      <plugin>
10.          <groupId>org.springframework.boot</groupId>
11.          <artifactId>spring-boot-maven-plugin</artifactId>
12.      </plugin>
13.  </plugins>
14. </build>

```

- les lignes 9-12 définissent le plugin qui va créer l'archive exécutable ;
- la ligne 3 définit la classe exécutable du projet ;

On procède ainsi :



- en [1] : on exécute une cible Maven ;
- en [2] : il y a deux cibles (goals) : [clean] pour supprimer le dossier [target] du projet Maven, [package] pour le régénérer ;
- en [3] : le dossier [target] généré, le sera dans ce dossier ;
- en [4] : on génère la cible ;

Dans les logs qui apparaissent dans la console, il est important de voir apparaître le plugin [**spring-boot-maven-plugin**]. C'est lui qui génère l'archive exécutable.

```
[INFO] --- spring-boot-maven-plugin:1.1.0.RELEASE:repackage (default) @ gs-rest-service ---
```

Avec une console, on se place dans le dossier généré :

```

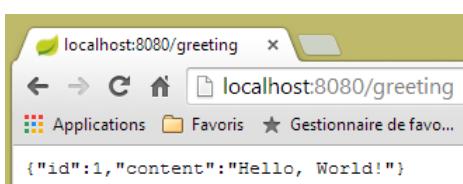
1. D:\Temp\wksSTS\gs-rest-service-complete\target>dir
2. ...
3. 11/06/2014  15:30    <DIR>          classes
4. 11/06/2014  15:30    <DIR>          generated-sources
5. 11/06/2014  15:30            11 073 572 gs-rest-service-0.1.0.jar
6. 11/06/2014  15:30            3 690 gs-rest-service-0.1.0.jar.original
7. 11/06/2014  15:30    <DIR>          maven-archiver
8. 11/06/2014  15:30    <DIR>          maven-status
9. ...

```

- ligne 5 : l'archive générée ;

Cette archive est exécutée de la façon suivante :

Maintenant que l'application web est lancée, on peut l'interroger avec un navigateur :



2.11.9 Déployer l'application sur un serveur Tomcat

Si Spring Boot s'avère très pratique en mode développement, il est probable qu'une application en production sera déployée sur un vrai serveur Tomcat. Voici comment procéder :

Modifier le fichier [pom.xml] de la façon suivante :

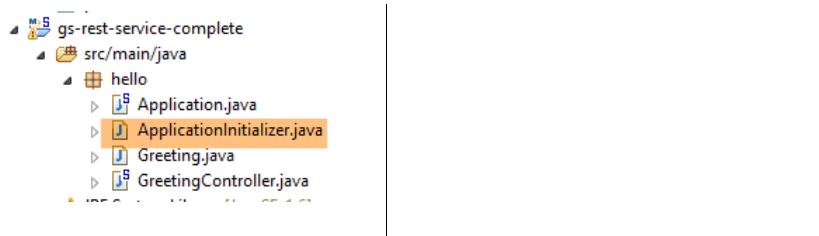
```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4.   <modelVersion>4.0.0</modelVersion>
5.
6.   <groupId>org.springframework</groupId>
7.   <artifactId>gs-rest-service</artifactId>
8.   <version>0.1.0</version>
9.   <packaging>war</packaging>
10.
11.  <parent>
12.    <groupId>org.springframework.boot</groupId>
13.    <artifactId>spring-boot-starter-parent</artifactId>
14.    <version>1.1.0.RELEASE</version>
15.  </parent>
16.
17.  <dependencies>
18.    <dependency>
19.      <groupId>org.springframework.boot</groupId>
20.      <artifactId>spring-boot-starter-web</artifactId>
21.    </dependency>
22.    <dependency>
23.      <groupId>com.fasterxml.jackson.core</groupId>
24.      <artifactId>jackson-databind</artifactId>
25.    </dependency>
26.    <dependency>
27.      <groupId>org.springframework.boot</groupId>
28.      <artifactId>spring-boot-starter-tomcat</artifactId>
29.      <scope>provided</scope>
30.    </dependency>
31.  </dependencies>
32.
33.  <properties>
34.    <start-class>hello.Application</start-class>
35.  </properties>
36. ...
```

37. </project>

Les modifications sont à faire à deux endroits :

- ligne 9 : il faut indiquer qu'on va générer une archive war (Web ARchive) ;
- lignes 26-30 : il faut ajouter une dépendance sur l'artifact [spring-boot-starter-tomcat]. Cet artifact amène toutes les classes de Tomcat dans les dépendances du projet ;
- ligne 29 : cet artifact est [provided], c-à-d que les archives correspondantes ne seront pas placées dans le war généré. En effet, ces archives seront trouvées sur le serveur Tomcat sur lequel s'exécutera l'application ;

Il faut par ailleurs configurer l'application web. En l'absence de fichier [web.xml], cela se fait avec une classe héritant de [SpringBootServletInitializer] :

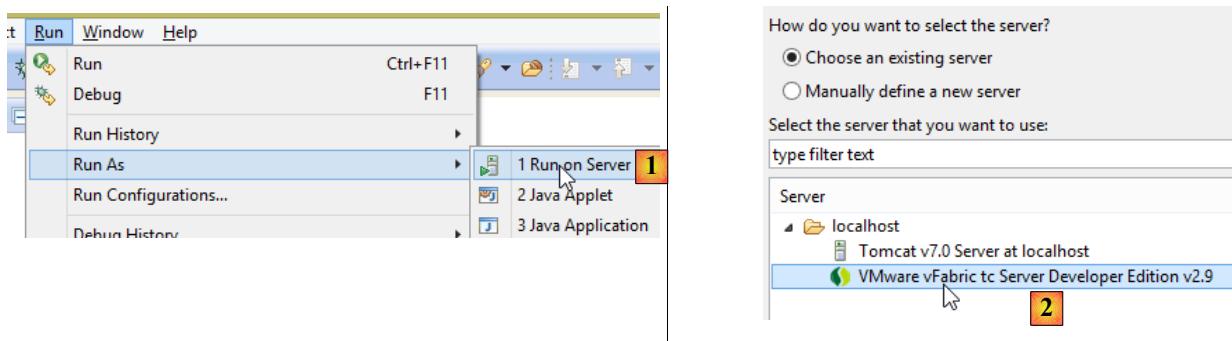


La classe [ApplicationInitializer] est la suivante :

```
1. package hello;
2.
3. import org.springframework.boot.builder.SpringApplicationBuilder;
4. import org.springframework.boot.context.web.SpringBootServletInitializer;
5.
6. public class ApplicationInitializer extends SpringBootServletInitializer {
7.
8.     @Override
9.     protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
10.         return application.sources(Application.class);
11.     }
12.
13. }
```

- ligne 6 : la classe [ApplicationInitializer] étend la classe [SpringBootServletInitializer] ;
- ligne 9 : la méthode [configure] est redéfinie (ligne 8) ;
- ligne 10 : on fournit la classe qui configure le projet ;

Pour exécuter le projet, on peut procéder ainsi :



- en [1], on exécute le projet sur l'un des serveurs enregistrés dans l'IDE Eclipse ;
- en [2], on choisit [tc Server Developer] qui est présent par défaut. C'est une variante de Tomcat ;

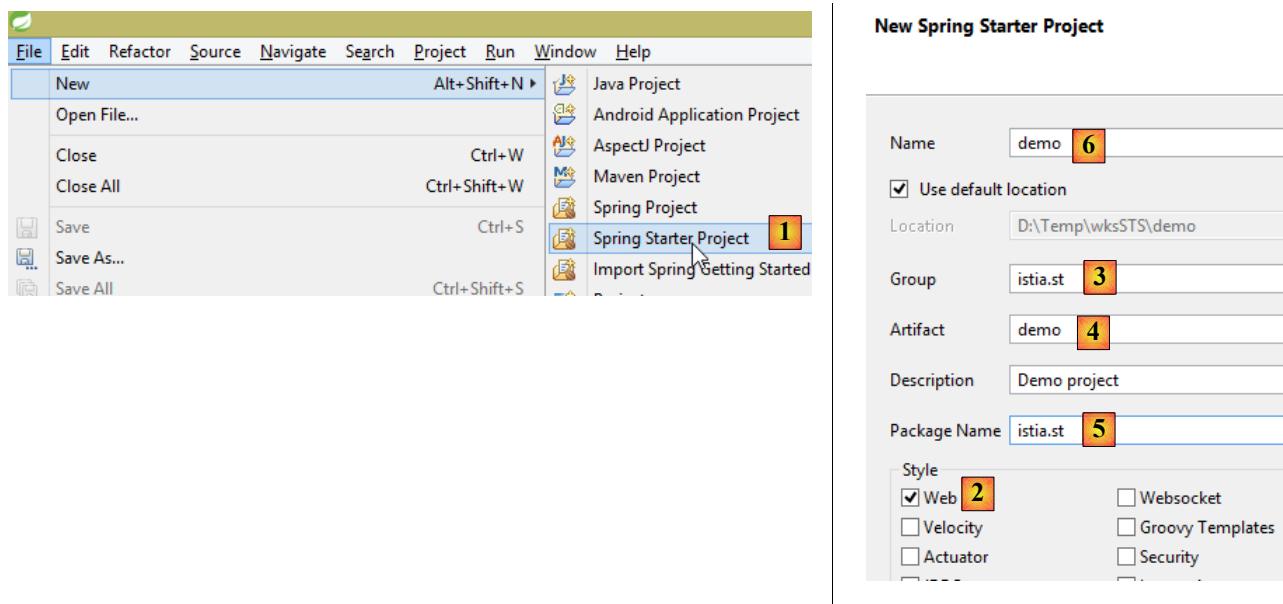
Ceci fait, on peut demander l'URL [<http://localhost:8080/gs-rest-service/greeting/?name=Mitchell>] dans un navigateur :



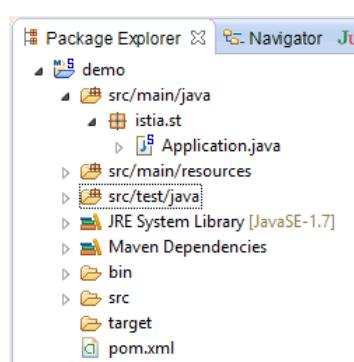
Nous savons désormais générer une archive war. Par la suite, nous continuerons à travailler avec Spring Boot et son archive jar exécutable.

2.11.10 Créer un nouveau projet web

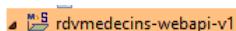
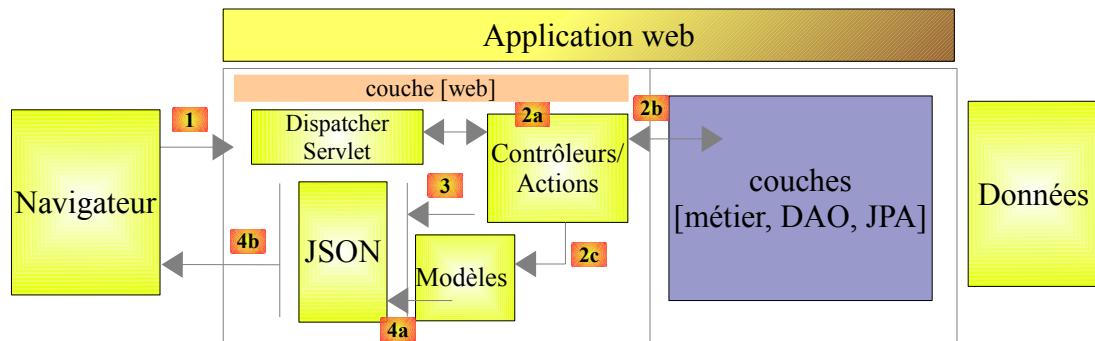
Pour construire un nouveau projet web, on peut procéder de la façon suivante :



- en [1] : File / New / Spring Starter Project
- en [2] : sélectionner [Web]. On ne sélectionne pas de bibliothèques de vues car dans un service web / JSON, il n'y a pas de vues ;
- le projet créé va être un projet Maven. En [3], on met le groupe de l'artifact Maven qui va être créé, en [4], le nom de l'artifact ;
- en [5], on met le nom d'un package où Spring va placer la classe de configuration du projet ;
- en [6], on donne un nom au projet Eclipse – peut être différent de [4] ;



2.12 La couche [web]



Nous allons construire la couche web en plusieurs étapes :

- étape 1 : une couche web opérationnelle sans authentification ;
- étape 2 : mise en place de l'authentification avec Spring Security ;
- étape 3 : mise en place des CORS [Cross-origin resource sharing (CORS) is a mechanism that allows many resources (e.g. fonts, JavaScript, etc.) on a web page to be requested from another domain outside the domain the resource originated from. (Wikipedia)]. Le client de notre service web sera un client web Angular qui n'appartiendra pas nécessairement au même domaine que notre service web. Par défaut, il ne peut alors pas y accéder sauf si le service web l'y autorise. Nous verrons comment ;

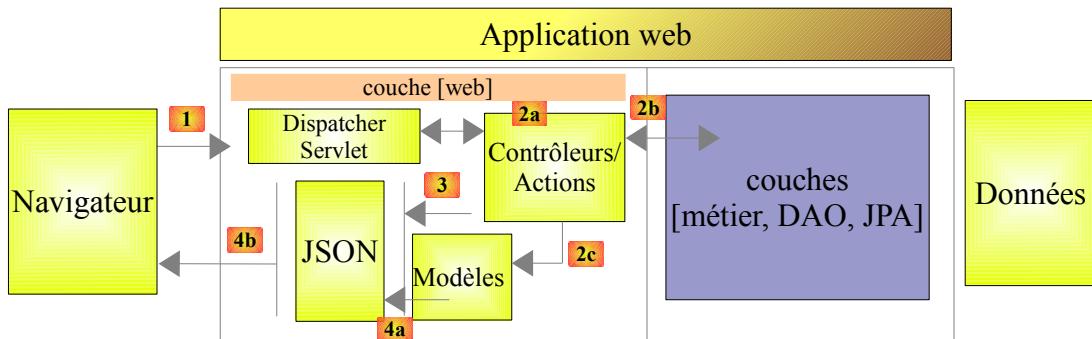
2.12.1 Configuration Maven

Le fichier [pom.xml] du projet est le suivant :

```
1. <modelVersion>4.0.0</modelVersion>
2. <groupId>istia.st.spring4.mvc</groupId>
3. <artifactId>rdvmedecins-webapi-v1</artifactId>
4. <version>0.0.1-SNAPSHOT</version>
5. <name>rdvmedecins-webapi-v1</name>
6. <description>Gestion de RV Médecins</description>
7. <parent>
8.   <groupId>org.springframework.boot</groupId>
9.   <artifactId>spring-boot-starter-parent</artifactId>
10.  <version>1.0.0.RELEASE</version>
11. </parent>
12. <dependencies>
13.   <dependency>
14.     <groupId>org.springframework.boot</groupId>
15.     <artifactId>spring-boot-starter-web</artifactId>
16.   </dependency>
17.   <dependency>
18.     <groupId>istia.st.spring4.rdvmedecins</groupId>
19.     <artifactId>rdvmedecins-metier-dao</artifactId>
20.     <version>0.0.1-SNAPSHOT</version>
21.   </dependency>
22. </dependencies>
```

- lignes 7-11 : le projet Maven parent ;
- lignes 13-16 : les dépendances pour un projet Spring MVC ;
- lignes 17-21 : les dépendances sur le projet des couches [métier, DAO, JPA] ;

2.12.2 L'interface du service web



- en [1], ci-dessus, le navigateur ne peut demander qu'un nombre restreint d'URL avec une syntaxe précise ;
- en [4], il reçoit une réponse JSON ;

Les réponses de notre service web auront toutes la même forme correspondant à la transformation JSON d'un objet de type [Reponse] suivant :

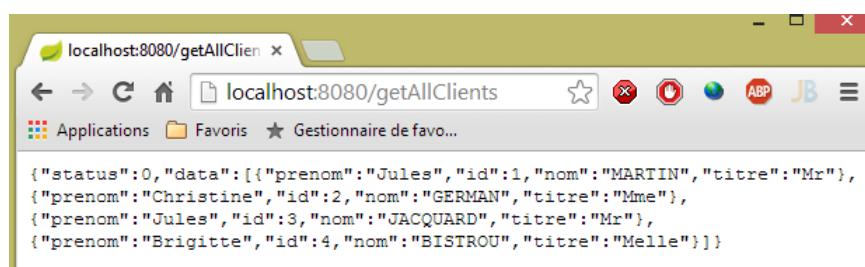
```

1. package rdvmedecins.web.models;
2.
3. public class Reponse {
4.
5.     // ----- propriétés
6.     // statut de l'opération
7.     private int status;
8.     // la réponse JSON
9.     private Object data;
10.
11.    // -----constructeurs
12.    public Reponse() {
13.    }
14.
15.    public Reponse(int status, Object data) {
16.        this.status = status;
17.        this.data = data;
18.    }
19.
20.    // méthodes
21.    public void incrStatusBy(int increment) {
22.        status += increment;
23.    }
24.
25.    // -----getters et setters
26. ...
27. }
```

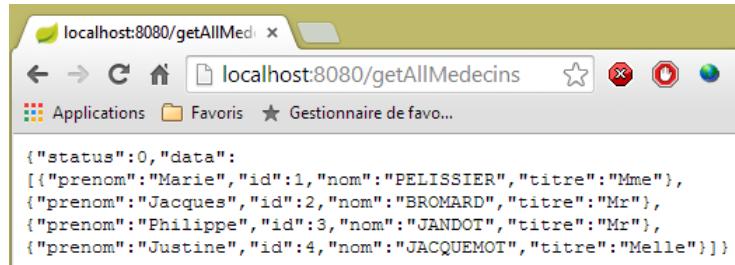
- ligne 7 : code d'erreur de la réponse 0: OK, autre chose : KO ;
- ligne 9 : le corps de la réponse ;

Nous présentons maintenant les copies d'écran qui illustrent l'interface du service web / JSON :

Liste de tous les patients du cabinet médical [/getAllClients]

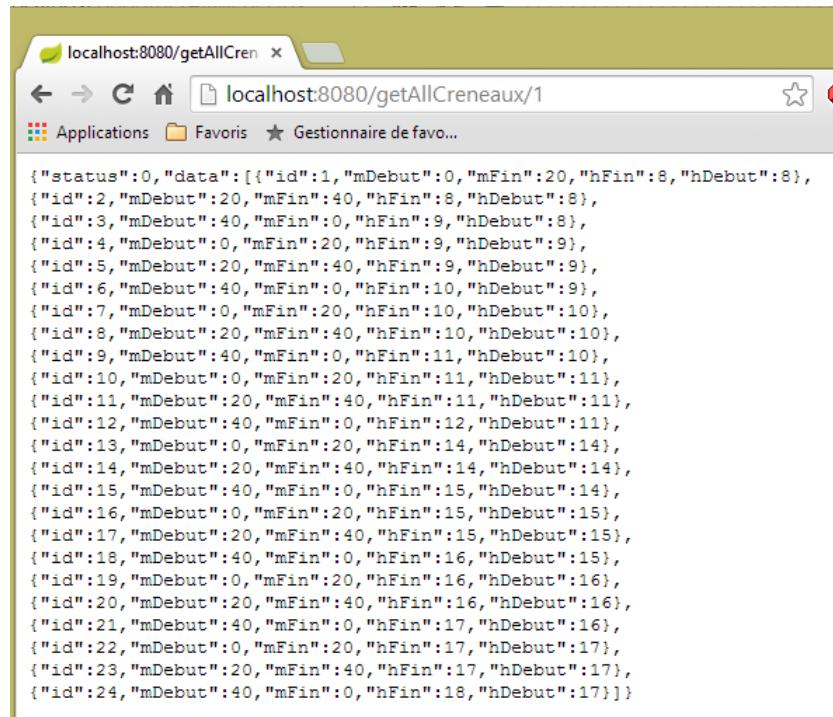


Liste de tous les médecins du cabinet médical [/getAllMedecins]



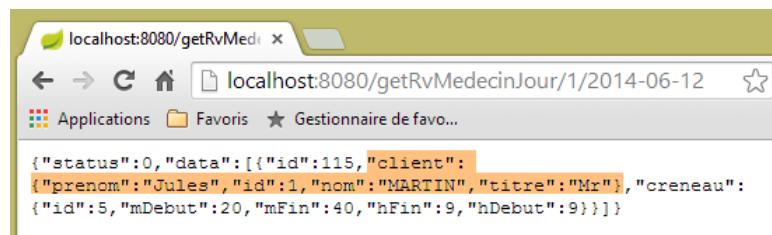
```
{"status":0,"data": [{"prenom":"Marie","id":1,"nom":"PELISSIER","titre":"Mme"}, {"prenom":"Jacques","id":2,"nom":"BROMARD","titre":"Mr"}, {"prenom":"Philippe","id":3,"nom":"JANDOT","titre":"Mr"}, {"prenom":"Justine","id":4,"nom":"JACQUEMOT","titre":"Melle"}]}
```

Liste des créneaux horaires d'un médecin [/getAllCreneaux/{idMedecin}]



```
{"status":0,"data": [{"id":1,"mDebut":0,"mFin":20,"hFin":8,"hDebut":8}, {"id":2,"mDebut":20,"mFin":40,"hFin":8,"hDebut":8}, {"id":3,"mDebut":40,"mFin":0,"hFin":9,"hDebut":8}, {"id":4,"mDebut":0,"mFin":20,"hFin":9,"hDebut":9}, {"id":5,"mDebut":20,"mFin":40,"hFin":9,"hDebut":9}, {"id":6,"mDebut":40,"mFin":0,"hFin":10,"hDebut":9}, {"id":7,"mDebut":0,"mFin":20,"hFin":10,"hDebut":10}, {"id":8,"mDebut":20,"mFin":40,"hFin":10,"hDebut":10}, {"id":9,"mDebut":40,"mFin":0,"hFin":11,"hDebut":10}, {"id":10,"mDebut":0,"mFin":20,"hFin":11,"hDebut":11}, {"id":11,"mDebut":20,"mFin":40,"hFin":11,"hDebut":11}, {"id":12,"mDebut":40,"mFin":0,"hFin":12,"hDebut":11}, {"id":13,"mDebut":0,"mFin":20,"hFin":14,"hDebut":14}, {"id":14,"mDebut":20,"mFin":40,"hFin":14,"hDebut":14}, {"id":15,"mDebut":40,"mFin":0,"hFin":15,"hDebut":14}, {"id":16,"mDebut":0,"mFin":20,"hFin":15,"hDebut":15}, {"id":17,"mDebut":20,"mFin":40,"hFin":15,"hDebut":15}, {"id":18,"mDebut":40,"mFin":0,"hFin":16,"hDebut":15}, {"id":19,"mDebut":0,"mFin":20,"hFin":16,"hDebut":16}, {"id":20,"mDebut":20,"mFin":40,"hFin":16,"hDebut":16}, {"id":21,"mDebut":40,"mFin":0,"hFin":17,"hDebut":16}, {"id":22,"mDebut":0,"mFin":20,"hFin":17,"hDebut":17}, {"id":23,"mDebut":20,"mFin":40,"hFin":17,"hDebut":17}, {"id":24,"mDebut":40,"mFin":0,"hFin":18,"hDebut":17}]}]
```

Liste des rendez-vous d'un médecin [/getRvMedecinJour/{idMedecin}/{aaaa-mm-jj}]



```
{"status":0,"data": [{"id":115,"client": {"prenom":"Jules","id":1,"nom":"MARTIN","titre":"Mr"}, "creneau": {"id":5,"mDebut":20,"mFin":40,"hFin":9,"hDebut":9}}]}
```

Agenda d'un médecin [/getAgendaMedecinJour/{idMedecin}/{aaaa-mm-jj}]

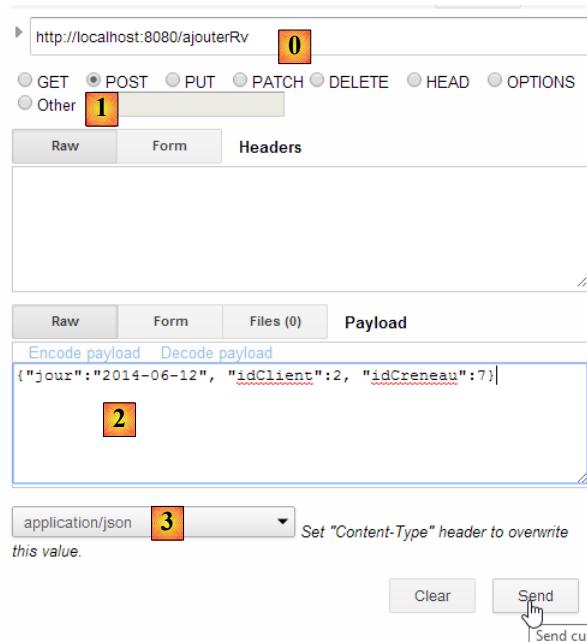
```

{
    "status": 0,
    "data": {
        "medecin": {
            "prenom": "Marie",
            "id": 1,
            "nom": "PELISSIER",
            "titre": "Mme"
        },
        "creneauxMedecin": [
            {"rv": null, "creneau": {"id": 1, "mDebut": 0, "mFin": 20, "hFin": 8, "hDebut": 8}}, {"rv": null, "creneau": {"id": 2, "mDebut": 20, "mFin": 40, "hFin": 8, "hDebut": 8}}, {"rv": null, "creneau": {"id": 3, "mDebut": 40, "mFin": 0, "hFin": 9, "hDebut": 8}}, {"rv": null, "creneau": {"id": 4, "mDebut": 0, "mFin": 20, "hFin": 9, "hDebut": 9}}, {"rv": {"id": 115, "client": {"prenom": "Jules", "id": 1, "nom": "MARTIN", "titre": "Mr"}}, "creneau": {"id": 5, "mDebut": 20, "mFin": 40, "hFin": 9, "hDebut": 9}}, {"creneau": {"id": 5, "mDebut": 20, "mFin": 40, "hFin": 9, "hDebut": 9}}, {"rv": null, "creneau": {"id": 7, "mDebut": 0, "mFin": 20, "hFin": 10, "hDebut": 10}}, {"rv": null, "creneau": {"id": 8, "mDebut": 20, "mFin": 40, "hFin": 10, "hDebut": 10}}, {"rv": null, "creneau": {"id": 9, "mDebut": 40, "mFin": 0, "hFin": 11, "hDebut": 10}}, {"rv": null, "creneau": {"id": 10, "mDebut": 0, "mFin": 20, "hFin": 11, "hDebut": 11}}, {"rv": null, "creneau": {"id": 11, "mDebut": 20, "mFin": 40, "hFin": 11, "hDebut": 11}}, {"rv": null, "creneau": {"id": 12, "mDebut": 40, "mFin": 0, "hFin": 12, "hDebut": 11}}, {"rv": null, "creneau": {"id": 13, "mDebut": 0, "mFin": 20, "hFin": 14, "hDebut": 14}}, {"rv": null, "creneau": {"id": 14, "mDebut": 20, "mFin": 40, "hFin": 14, "hDebut": 14}}, {"rv": null, "creneau": {"id": 15, "mDebut": 40, "mFin": 0, "hFin": 15, "hDebut": 14}}, {"rv": null, "creneau": {"id": 16, "mDebut": 0, "mFin": 20, "hFin": 15, "hDebut": 15}}, {"rv": null, "creneau": {"id": 17, "mDebut": 20, "mFin": 40, "hFin": 15, "hDebut": 15}}, {"rv": null, "creneau": {"id": 18, "mDebut": 40, "mFin": 0, "hFin": 16, "hDebut": 15}}, {"rv": null, "creneau": {"id": 19, "mDebut": 0, "mFin": 20, "hFin": 16, "hDebut": 16}}, {"rv": null, "creneau": {"id": 20, "mDebut": 20, "mFin": 40, "hFin": 16, "hDebut": 16}}, {"rv": null, "creneau": {"id": 21, "mDebut": 40, "mFin": 0, "hFin": 17, "hDebut": 16}}, {"rv": null, "creneau": {"id": 22, "mDebut": 0, "mFin": 20, "hFin": 17, "hDebut": 17}}, {"rv": null, "creneau": {"id": 23, "mDebut": 20, "mFin": 40, "hFin": 17, "hDebut": 17}}, {"rv": null, "creneau": {"id": 24, "mDebut": 40, "mFin": 0, "hFin": 18, "hDebut": 17}}], "jour": "2014-06-12"
}

```

Pour ajouter / supprimer un rendez-vous nous utilisons le complément Chrome [Advanced Rest Client] car ces opérations se font avec un POST.

Ajouter un rendez-vous [/ajouterRv]



- en [0], l'URL du service web ;
- en [1], la méthode POST est utilisée ;
- en [2], le texte JSON des informations transmises au service web sous la forme `{jour, idClient, idCreneau}` ;
- en [3], le client précise au service web qu'il lui envoie des informations au format JSON ;

La réponse est alors la suivante :

Status 200 OK ⏱ Loading time: 136 ms

Request headers

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.152 Safari/537.36
Origin: chrome-extension://hgmloofddfdnphfgcellkdfbfjelo
Content-Type: application/json 4
Accept: */*
Accept-Encoding: gzip,deflate,sdch
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

Response headers

Server: Apache-Coyote/1.1 ⏱
Content-Type: application/json; charset=UTF-8 5
Transfer-Encoding: chunked ⏱
Date: Thu, 12 Jun 2014 09:28:51 GMT ⏱

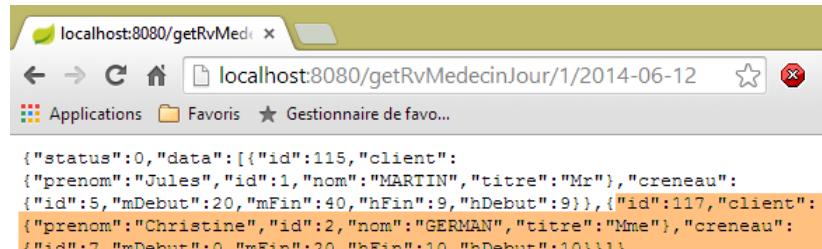
Raw JSON Response

Copy to clipboard Save as file

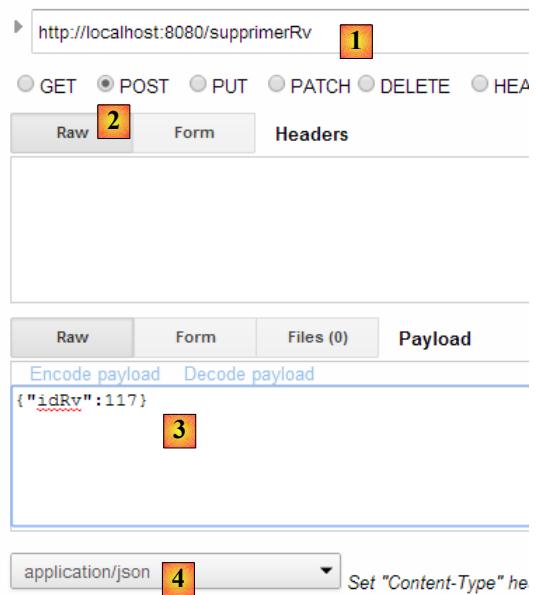
```
{  
    status: 0  
    -data: {  
        id: 117 6  
        -client: {  
            prenom: "Christine"  
            id: 2  
            nom: "GERMAN"  
            titre: "Mme"  
        }  
        -creneau: {  
            id: 7  
            mDebut: 0  
            mFin: 20  
            hFin: 10  
            hDebut: 10  
        }  
    }  
}
```

- en [4] : le client envoie l'entête signifiant que les données qu'il envoie sont au format JSON ;
- en [5] : le service web répond qu'il envoie lui aussi du JSON ;
- en [6] : la réponse JSON du service web. Le champ [data] contient la forme JSON du rendez-vous ajouté ;

La présence du nouveau rendez-vous peut être vérifié :



Supprimer un rendez-vous [/supprimerRv]



- en [1], l'URL du service web ;
 - en [2], la méthode POST est utilisée;
 - en [3], le texte JSON des informations transmises au service web sous la forme {idRv} ;
 - en [4], le client précise au service web qu'il lui envoie des informations JSON ;

La réponse est alors la suivante :

Status	200 OK	Loading time: 117 ms
Request headers	User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2049.0 Safari/537.36 Origin: chrome-extension://hgmloofddffndpobfkcddepelkddigk Content-Type: application/json Accept: */* Accept-Encoding: gzip,deflate,sdch Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6	
Response headers	Server: Apache-Coyote/1.1 Content-Type: application/json;charset=UTF-8 Transfer-Encoding: chunked Date: Thu, 12 Jun 2014 09:50:22 GMT	

- en [5] : le champ [status] est à 0, montrant par là que l'opération a réussi ;

La suppression du rendez-vous peut être vérifiée :

A screenshot of a web browser window titled "localhost:8080/getRvMedecinJour/1/2014-06-12". The URL bar shows the same address. The page content displays a JSON object:

```
{"status":0,"data":[{"id":115,"client":{"prenom":"Jules","id":1,"nom":"MARTIN","titre":"Mr"},"creneau":{"id":5,"mDebut":20,"mFin":40,"hFin":9,"hDebut":9}}]}
```

Ci-dessus, le rendez-vous du patient [Mme GERMAN] n'est plus présent.

Le service web permet également de récupérer des entités via leur identifiant :

A screenshot of a web browser window titled "localhost:8080/getClientById/1". The URL bar shows the same address. The page content displays a JSON object:

```
{"status":0,"data":{"prenom":"Jules","id":1,"nom":"MARTIN","titre":"Mr"}}
```

A screenshot of a web browser window titled "localhost:8080/getMedecinById/2". The URL bar shows the same address. The page content displays a JSON object:

```
{"status":0,"data":{"prenom":"Jacques","id":2,"nom":"BROMARD","titre":"Mr"}}
```

A screenshot of a web browser window titled "localhost:8080/getCreneauById/7". The URL bar shows the same address. The page content displays a JSON object:

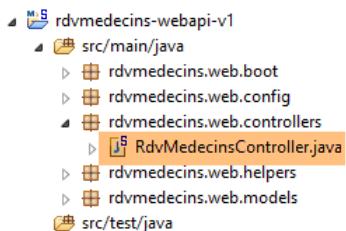
```
{"status":0,"data":{"id":7,"mDebut":0,"mFin":20,"hFin":10,"hDebut":10}}
```

A screenshot of a web browser window titled "localhost:8080/getRvById/115". The URL bar shows the same address. The page content displays a JSON object:

```
{"status":0,"data":{"id":115,"idClient":1,"idCreneau":5}}
```

Toutes ces URLs sont traitées par le contrôleur [RdvMedecinsController] que nous présentons maintenant.

2.12.3 Le squelette du contrôleur [RdvMedecinsController]



Le contrôleur [RdvMedecinsController] est le suivant :

```
1. package rdvmedecins.web.controllers;
2.
3. import java.text.ParseException;
4. ...
5.
6. @RestController
7. public class RdvMedecinsController {
8.
9.     @Autowired
10.    private ApplicationModel application;
11.    private List<String> messages;
12.
13.    @PostConstruct
14.    public void init() {
15.        // messages d'erreur de l'application
16.        messages = application.getMessages();
17.    }
18.
19.    // liste des médecins
20.    @RequestMapping(value = "/getAllMedecins", method = RequestMethod.GET)
21.    public Reponse getAllMedecins() {
22.    ...
23.    }
24.
25.    // liste des clients
26.    @RequestMapping(value = "/getAllClients", method = RequestMethod.GET)
27.    public Reponse getAllClients() {
28.    ...
29.    }
30.
31.    // liste des créneaux d'un médecin
32.    @RequestMapping(value = "/getAllCreneaux/{idMedecin}", method = RequestMethod.GET)
33.    public Reponse getAllCreneaux(@PathVariable("idMedecin") long idMedecin) {
34.    ...
35.    }
36.
37.    // liste des rendez-vous d'un médecin
38.    @RequestMapping(value = "/getRvMedecinJour/{idMedecin}/{jour}", method = RequestMethod.GET)
39.    public Reponse getRvMedecinJour(@PathVariable("idMedecin") long idMedecin,
40.                                    @PathVariable("jour") String jour) {
41.    ...
42.    }
43.
44.    @RequestMapping(value = "/getClientById/{id}", method = RequestMethod.GET)
45.    public Reponse getClientById(@PathVariable("id") long id) {
46.    ...
47.    }
48.
49.    @RequestMapping(value = "/getMedecinById/{id}", method = RequestMethod.GET)
50.    public Reponse getMedecinById(@PathVariable("id") long id) {
51.    ...
52.    }
53.
54.    @RequestMapping(value = "/getRvById/{id}", method = RequestMethod.GET)
55.    public Reponse getRvById(@PathVariable("id") long id) {
56.    ...
57.    }
```

```

58.
59.     @RequestMapping(value = "/getCreneauById/{id}", method = RequestMethod.GET)
60.     public Reponse getCreneauById(@PathVariable("id") long id) {
61. ...
62. }
63.
64.     @RequestMapping(value = "/ajouterRv", method = RequestMethod.POST, consumes = "application/json;
charset=UTF-8")
65.     public Reponse ajouterRv(@RequestBody PostAjouterRv post) {
66. ...
67. }
68.
69.     @RequestMapping(value = "/supprimerRv", method = RequestMethod.POST, consumes = "application/json;
charset=UTF-8")
70.     public Reponse supprimerRv(@RequestBody PostSupprimerRv post) {
71. ...
72. }
73.
74.     @RequestMapping(value = "/getAgendaMedecinJour/{idMedecin}/{jour}", method = RequestMethod.GET)
75.     public Reponse getAgendaMedecinJour(
76.         @PathVariable("idMedecin") long idMedecin,
77.         @PathVariable("jour") String jour) {
78. ...
79. }
80. }
```

- ligne 6 : l'annotation **[@RestController]** fait de la classe [RdvMedecinsController] un contrôleur Spring. Par ailleurs, elle entraîne également que les méthodes traitant les URL vont générer une réponse qui sera automatiquement transformée en JSON ;
- lignes 9-10 : un objet de type [ApplicationModel] sera injecté ici par Spring ;
- ligne 13 : l'annotation **[@PostConstruct]** tague une méthode à exécuter juste après l'instanciation de la classe. Lorsqu'elle celle-ci s'exécute, les objets injectés par Spring sont disponibles ;
- toutes les méthodes rendent un objet de type [Reponse] suivant :

```

1. package rdvmedecins.web.models;
2.
3. public class Reponse {
4.
5.     // ----- propriétés
6.     // statut de l'opération
7.     private int status;
8.     // la réponse
9.     private Object data;
10. ...
11. }
```

Cet objet est sérialisé en JSON avant d'être envoyé au navigateur client ;

- ligne 20 : l'annotation **[@RequestMapping]** fixe les conditions d'appel de la méthode. Ici la méthode traite une demande GET de l'URL [/getAllMedecins]. Si cette URL était demandée par un POST, elle serait refusée et Spring MVC enverrait un code HTTP d'erreur au client web ;
- ligne 32 : l'URL est paramétrée par {idMedecin}. Ce paramètre est récupéré avec l'annotation **[@PathVariable]** ligne 33 ;
- ligne 33 : l'unique paramètre [**long** idMedecin] reçoit sa valeur du paramètre {idMedecin} de l'URL **[@PathVariable("idMedecin")]**. Le paramètre dans l'URL et celui de la méthode peuvent porter des noms différents. Il faut noter ici que **[@PathVariable("idMedecin")]** est de type String (toute l'URL est un String) alors que le paramètre [**long** idMedecin] est de type [long]. Le changement de type est fait automatiquement. Un code d'erreur HTTP est renvoyé si ce changement de type échoue ;
- ligne 65 : l'annotation **[@RequestBody]** désigne le corps de la requête. Dans une requête GET, il n'y a quasiment jamais de corps (mais il est possible d'en mettre un). Dans une requête POST, il y en a le plus souvent (mais il est possible de ne pas en mettre). Pour l'URL [ajouterRv], le client web envoie dans son POST la chaîne JSON suivante :

```
{"jour": "2014-06-12", "idClient": 3, "idCreneau": 7}
```

La syntaxe **[@RequestBody PostAjouterRv post]** (ligne 65) ajoutée au fait que la méthode attend du JSON [consumes = "application/json; charset=UTF-8"] ligne 64 va faire que la chaîne JSON envoyée par le client web va être déserialisée en un objet de type [PostAjouter]. Celui-ci est le suivant :

```

1. package rdvmedecins.web.models;
2.
```

```

3. public class PostAjouterRv {
4.
5.     // données du post
6.     private String jour;
7.     private long idClient;
8.     private long idCreneau;
9.
10.    // getters et setters
11.    ...
12.}
```

Là également, les changements de type nécessaires auront lieu automatiquement ;

- lignes 69-70, on trouve un mécanisme similaire pour l'URL [/supprimerRv]. La chaîne JSON postée est la suivante :

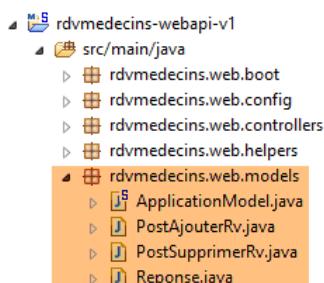
```
{"idRv":116}
```

et le type [PostSupprimerRv] le suivant :

```

1. package rdvmedecins.web.models;
2.
3. public class PostSupprimerRv {
4.
5.     // données du post
6.     private long idRv;
7.
8.     // getters et setters
9.     ...
10.}
```

2.12.4 Les modèles du service web



Nous avons déjà présenté les modèles [Reponse, PostAjouterRv, PostSupprimerRv]. Le modèle [ApplicationModel] est le suivant :

```

1. package rdvmedecins.web.models;
2.
3. import java.util.Date;
4. ...
5.
6. @Component
7. public class ApplicationModel implements IMetier {
8.
9.     // la couche [métier]
10.    @Autowired
11.    private IMetier métier;
12.
13.    // données provenant de la couche [métier]
14.    private List<Medecin> médecins;
15.    private List<Client> clients;
16.    // messages d'erreur
17.    private List<String> messages;
18.
19.    @PostConstruct
20.    public void init() {
21.        // on récupère les médecins et les clients
22.        try {
23.            médecins = métier.getAllMedecins();
24.            clients = métier.getAllClients();
```

```

25.     } catch (Exception ex) {
26.         messages = Static.getErreursForException(ex);
27.     }
28. }
29.
30. // getter
31. public List<String> getMessages() {
32.     return messages;
33. }
34.
35. // ----- interface couche [métier]
36. @Override
37. public List<Client> getAllClients() {
38.     return clients;
39. }
40.
41. @Override
42. public List<Medecin> getAllMedecins() {
43.     return medecins;
44. }
45.
46. @Override
47. public List<Creneau> getAllCreneaux(long idMedecin) {
48.     return métier.getAllCreneaux(idMedecin);
49. }
50.
51. @Override
52. public List<Rv> getRvMedecinJour(long idMedecin, Date jour) {
53.     return métier.getRvMedecinJour(idMedecin, jour);
54. }
55.
56. @Override
57. public Client getClientById(long id) {
58.     return métier.getClientById(id);
59. }
60.
61. @Override
62. public Medecin getMedecinById(long id) {
63.     return métier.getMedecinById(id);
64. }
65.
66. @Override
67. public Rv getRvById(long id) {
68.     return métier.getRvById(id);
69. }
70.
71. @Override
72. public Creneau getCreneauById(long id) {
73.     return métier.getCreneauById(id);
74. }
75.
76. @Override
77. public Rv ajouterRv(Date jour, Creneau creneau, Client client) {
78.     return métier.ajouterRv(jour, creneau, client);
79. }
80.
81. @Override
82. public void supprimerRv(Rv rv) {
83.     métier.supprimerRv(rv);
84. }
85.
86. @Override
87. public AgendaMedecinJour getAgendaMedecinJour(long idMedecin, Date jour) {
88.     return métier.getAgendaMedecinJour(idMedecin, jour);
89. }
90.
91. }

```

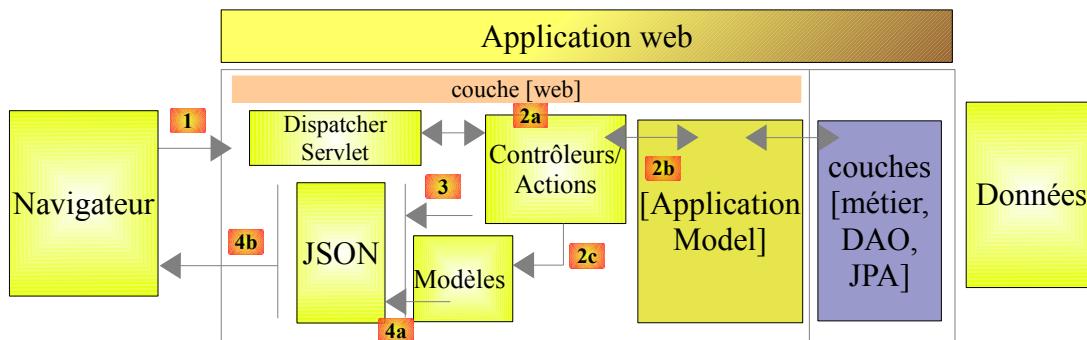
- ligne 6 : l'annotation **[@Component]** fait de la classe **[ApplicationModel]** un composant Spring. Comme tous les composants Spring vus jusqu'ici (à l'exception de **@Controller**), un seul objet de ce type sera instancié (**singleton**) ;
- ligne 7 : la classe **[ApplicationModel]** implémente l'interface **[IMetier]** ;
- lignes 10-11 : une référence sur la couche **[métier]** est injectée par Spring ;

- ligne 19 : l'annotation `@PostConstruct` fait que la méthode `[init]` va être exécutée juste après l'instanciation de la classe `[ApplicationModel]` ;
- lignes 23-24 : on récupère les listes de médecins et de clients auprès de la couche `[métier]` ;
- ligne 26 : si une exception se produit, on stocke les messages de la pile d'exceptions dans le champ de la ligne 17 ;

La classe `[ApplicationModel]` va nous servir à deux choses :

- de cache pour stocker les listes de médecins et de patients (clients) ;
- d'interface unique pour les contrôleurs ;

L'architecture de la couche web évolue comme suit :

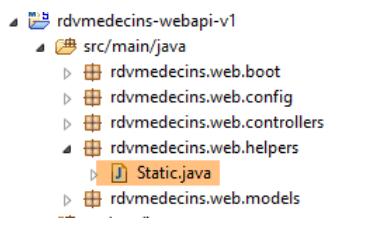


- en [2b], les méthodes du ou des contrôleurs communiquent avec le singleton `[ApplicationModel]` ;

Cette stratégie amène de la souplesse quant à la gestion du cache. Actuellement les créneaux horaires des médecins ne sont pas mis en cache. Pour les y mettre, il suffit de modifier la classe `[ApplicationModel]`. Cela n'a aucun impact sur le contrôleur qui continuera à utiliser la méthode `[List<Creneau> getAllCreneaux(long idMedecin)]` comme il le faisait auparavant. C'est l'implémentation de cette méthode dans `[ApplicationModel]` qui sera changée.

2.12.5 La classe Static

La classe `[Static]` regroupe un ensemble de méthodes statiques utilitaires qui n'ont pas d'aspect " métier " ou " web " :



Son code est le suivant :

```

1. package rdvmedecins.web.helpers;
2.
3. import java.text.SimpleDateFormat;
4. ...
5.
6. public class Static {
7.
8.     public Static() {
9.     }
10.
11.    // liste des messages d'erreur d'une exception
12.    public static List<String> getErreursForException(Exception exception) {
13.        // on récupère la liste des messages d'erreur de l'exception
14.        Throwable cause = exception;
15.        List<String> erreurs = new ArrayList<String>();
16.        while (cause != null) {
17.            erreurs.add(cause.getMessage());
18.            cause = cause.getCause();
19.        }
    
```

```

20.     return erreurs;
21. }
22.
23. // mappers Object --> Map
24. // -----
25. ....
26. }
```

- ligne 12 : la méthode [Static.getErreursForException] qui a été utilisée (ligne 8 ci-dessous) dans la méthode [init] de la classe [ApplicationModel] :

```

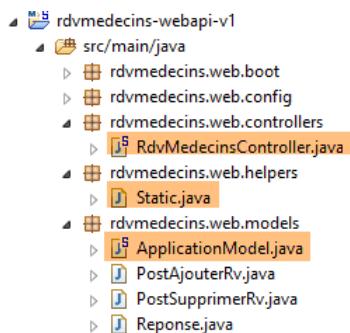
1. @PostConstruct
2. public void init() {
3.     // on récupère les médecins et les clients
4.     try {
5.         médecins = métier.getAllMedecins();
6.         clients = métier.getAllClients();
7.     } catch (Exception ex) {
8.         messages = Static.getErreursForException(ex);
9.     }
10. }
```

La méthode construit un objet [List<String>] avec les messages d'erreur [exception.getMessage()] d'une exception [exception] et de celles qu'elle contient [exception.getCause()].

La classe [Static] contient d'autres méthodes utilitaires sur lesquelles nous reviendrons lorsque nous les rencontrerons.

Nous allons maintenant détailler le traitement des URL du service web. Trois classes principales sont en jeu dans ce traitement :

- le contrôleur [**RdvMedecinsController**] ;
- la classe de méthodes utilitaires [**Static**] ;
- la classe de cache [**ApplicationModel**] ;



2.12.6 La méthode [init] du contrôleur

Le contrôleur [RdvMedecinsController] (cf page 78) a une méthode [init] qui est exécutée juste après son instantiation :

```

1. @Autowired
2. private ApplicationModel application;
3. private List<String> messages;
4.
5. @PostConstruct
6. public void init() {
7.     // messages d'erreur de l'application
8.     messages = application.getMessages();
9. }
```

- ligne 8 : les messages d'erreur stockés dans l'application cache [ApplicationModel] sont mémorisés en local dans le champ de la ligne 3. Cela va permettre aux méthodes de savoir si l'application s'est initialisée correctement.

2.12.7 L'URL [/getAllMedecins]

L'URL [/getAllMedecins] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

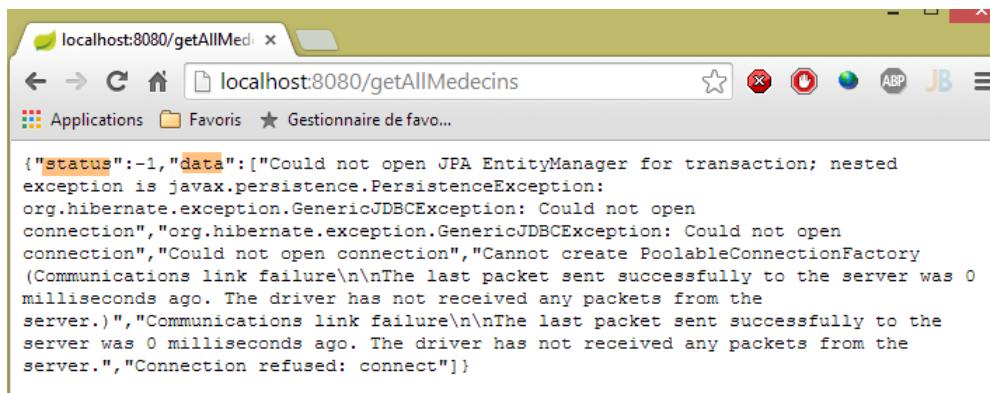
```

1.    // liste des médecins
2.    @RequestMapping(value = "/getAllMedecins", method = RequestMethod.GET)
3.    public Reponse getAllMedecins() {
4.        // état de l'application
5.        if (messages != null) {
6.            return new Reponse(-1, messages);
7.        }
8.        // liste des médecins
9.        try {
10.            return new Reponse(0, application.getAllMedecins());
11.        } catch (Exception e) {
12.            return new Reponse(1, Static.getErreursForException(e));
13.        }
14.    }

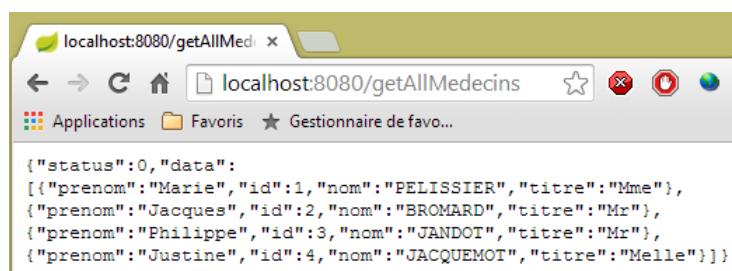
```

- ligne 5 : on regarde si l'application s'est correctement initialisée (messages==null). Si ce n'est pas le cas, on renvoie une réponse avec *status*=-1 et *data*=*messages* ;
- ligne 10 : sinon on renvoie la liste des médecins avec un *status* égal à 0. La méthode [application.getAllMedecins()] ne lance pas d'exception car elle se contente de rendre une liste qui est en cache. Néanmoins on gardera cette gestion d'exception pour le cas où les médecins ne seraient plus mis en cache ;

Nous n'avons pas encore illustré le cas où l'application s'est mal initialisée. Arrêtons le SGBD MySQL5, lançons le service web puis demandons l'URL [/getAllMedecins] :



On obtient bien une erreur. Dans un contexte normal, on obtient la vue suivante :



2.12.8 L'URL [/getAllClients]

L'URL [/getAllClients] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```

1.    // liste des clients
2.    @RequestMapping(value = "/getAllClients")
3.    public Reponse getAllClients() {
4.        // état de l'application
5.        if (messages != null) {
6.            return new Reponse(-1, messages);
7.        }

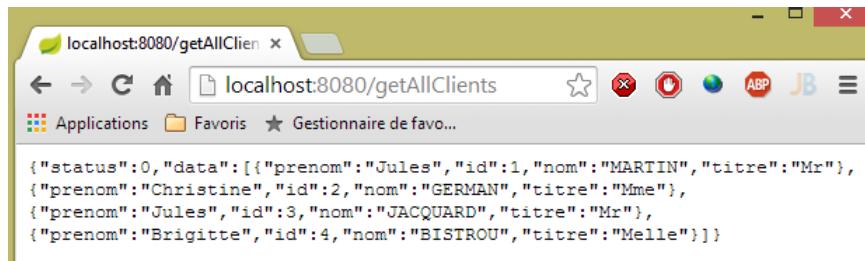
```

```

8.     // liste des clients
9.     try {
10.         return new Reponse(0, application.getAllClients());
11.     } catch (Exception e) {
12.         return new Reponse(1, Static.getErreursForException(e));
13.     }
14. }

```

Elle est analogue à la méthode [getAllMedecins] déjà étudiée. Les résultats obtenus sont les suivants :



2.12.9 L'URL [/getAllCreneaux/{idMedecin}]

L'URL [/getAllCreneaux/{idMedecin}] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```

1. // liste des créneaux d'un médecin
2. @RequestMapping(value = "/getAllCreneaux/{idMedecin}", method = RequestMethod.GET)
3. public Reponse getAllCreneaux(@PathVariable("idMedecin") long idMedecin) {
4.     // état de l'application
5.     if (messages != null) {
6.         return new Reponse(-1, messages);
7.     }
8.     // on récupère le médecin
9.     Reponse réponse = getMedecin(idMedecin);
10.    if (réponse.getStatus() != 0) {
11.        return réponse;
12.    }
13.    Medecin médecin = (Medecin) réponse.getData();
14.    // créneaux du médecin
15.    List<Creneau> créneaux = null;
16.    try {
17.        créneaux = application.getAllCreneaux(médecin.getId());
18.    } catch (Exception e1) {
19.        return new Reponse(3, Static.getErreursForException(e1));
20.    }
21.    // on rend la réponse
22.    return new Reponse(0, Static.getListMapForCreneaux(créneaux));
23. }

```

- ligne 9 : le médecin identifié par le paramètre [id] est demandé à une méthode locale :

```

1. private Reponse getMedecin(long id) {
2.     // on récupère le médecin
3.     Medecin médecin = null;
4.     try {
5.         médecin = application.getMedecinById(id);
6.     } catch (Exception e1) {
7.         return new Reponse(1, Static.getErreursForException(e1));
8.     }
9.     // médecin existant ?
10.    if (médecin == null) {
11.        return new Reponse(2, null);
12.    }
13.    // ok
14.    return new Reponse(0, médecin);
15. }

```

On revient de cette méthode avec un *status* dans [0,1,2]. Revenons au code de la méthode [getAllCreneaux] :

- lignes 10-12 : si `status!=0`, on rend immédiatement la réponse ;
- ligne 13 : on récupère le médecin ;
- ligne 17 : on récupère les créneaux de ce médecin ;
- ligne 22 : on envoie comme réponse un objet [`Static.getListMapForCreneaux(créneaux)`] ;

Rappelons la définition de la classe [Creneau] :

```

1.  @Entity
2.  @Table(name = "creneaux")
3.  public class Creneau extends AbstractEntity {
4.
5.      private static final long serialVersionUID = 1L;
6.      // caractéristiques d'un créneau de RV
7.      private int hdebut;
8.      private int mdebut;
9.      private int hfin;
10.     private int mfin;
11.
12.     // un créneau est lié à un médecin
13.     @ManyToOne(fetch = FetchType.LAZY)
14.     @JoinColumn(name = "id_medecin")
15.     private Medecin medecin;
16.
17.     // clé étrangère
18.     @Column(name = "id_medecin", insertable = false, updatable = false)
19.     private long idMedecin;
20. ...
21. }
```

- ligne 13 : le médecin est cherché en mode [FetchType.LAZY] ;

Rappelons la requête JPQL qui implémente la méthode [getAllCreneaux] dans la couche [DAO] :

```
@Query("select c from Creneau c where c.medecin.id=?1")
```

La notation [c.medecin.id] force la jointure entre les tables [CRENEAUX] et [MEDECINS]. Aussi la requête ramène-t-elle tous les créneaux du médecin avec dans chacun d'eux le médecin. Lorsqu'on sérialise en JSON ces créneaux, on voit apparaître la chaîne JSON du médecin dans chacun d'eux. C'est inutile. Aussi plutôt que de sérialiser un objet [Creneau], on va sérialiser un objet [Map] dans lequel on ne mettra que les champs désirés.

Revenons au code étudié initialement :

```

1.  // on rend la réponse
2.  return new Reponse(0, Static.getListMapForCreneaux(créneaux));
```

La méthode [Static.getListMapForCreneaux] est la suivante :

```

1.  // List<Creneau> --> List<Map>
2.  public static List<Map<String, Object>> getListMapForCreneaux(List<Creneau> créneaux) {
3.      // liste de dictionnaires <String, Object>
4.      List<Map<String, Object>> liste = new ArrayList<Map<String, Object>>();
5.      for (Creneau créneau : créneaux) {
6.          liste.add(Static.getMapForCreneau(créneau));
7.      }
8.      // on rend la liste
9.      return liste;
10. }
```

et la méthode [Static.getMapForCreneau] est la suivante :

```

1.  // Creneau --> Map
2.  public static Map<String, Object> getMapForCreneau(Creneau créneau) {
3.      // qq chose à faire ?
4.      if (créneau == null) {
5.          return null;
6.      }
7.      // dictionnaire <String, Object>
8.      Map<String, Object> hash = new HashMap<String, Object>();
```

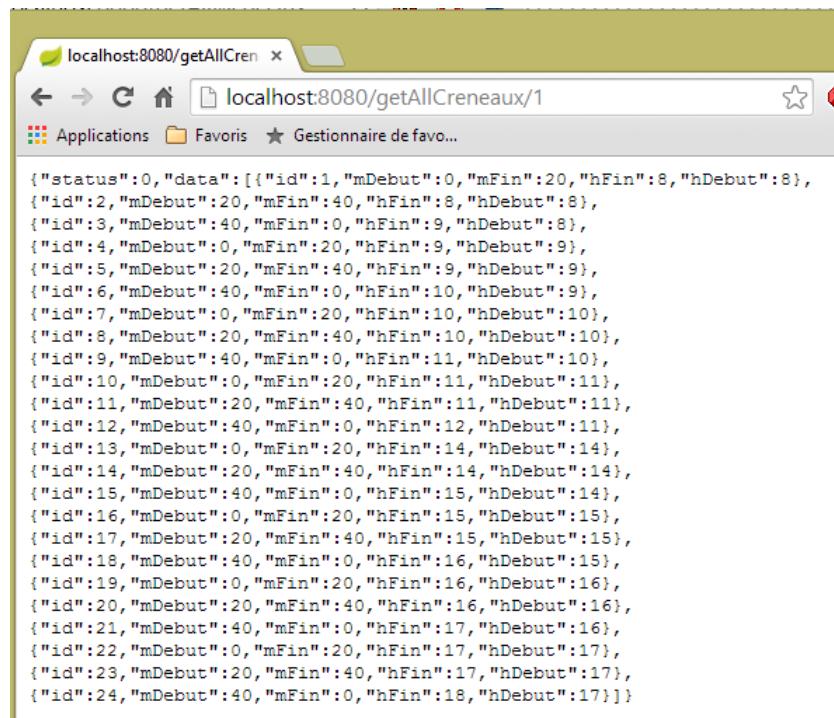
```

9.     hash.put("id", créneau.getId());
10.    hash.put("hDebut", créneau.getHdebut());
11.    hash.put("mDebut", créneau.getMdebut());
12.    hash.put("hFin", créneau.getHfin());
13.    hash.put("mFin", créneau.getMfin());
14.    // on rend le dictionnaire
15.    return hash;
16. }

```

- ligne 8 : on crée un dictionnaire ;
- lignes 9-13 : on y met les champs qu'on veut garder dans la chaîne JSON. Le champ [medecin] n'y est pas ;
- ligne 15 : on rend ce dictionnaire ;

Les résultats obtenus sont les suivants :



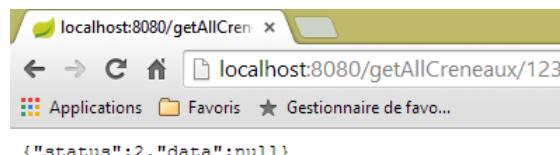
The screenshot shows a browser window with the URL `localhost:8080/getAllCreneaux/1`. The page content displays a large JSON array of objects, each representing a time slot (creneau) with fields `mDebut`, `mFin`, `hDebut`, and `hFin`.

```

{
  "status": 0,
  "data": [
    {"id": 1, "mDebut": 0, "mFin": 20, "hDebut": 8, "hFin": 8},
    {"id": 2, "mDebut": 20, "mFin": 40, "hDebut": 8, "hFin": 8},
    {"id": 3, "mDebut": 40, "mFin": 0, "hDebut": 9, "hFin": 9},
    {"id": 4, "mDebut": 0, "mFin": 20, "hDebut": 9, "hFin": 9},
    {"id": 5, "mDebut": 20, "mFin": 40, "hDebut": 9, "hFin": 9},
    {"id": 6, "mDebut": 40, "mFin": 0, "hDebut": 10, "hFin": 10},
    {"id": 7, "mDebut": 0, "mFin": 20, "hDebut": 10, "hFin": 10},
    {"id": 8, "mDebut": 20, "mFin": 40, "hDebut": 10, "hFin": 10},
    {"id": 9, "mDebut": 40, "mFin": 0, "hDebut": 11, "hFin": 11},
    {"id": 10, "mDebut": 0, "mFin": 20, "hDebut": 11, "hFin": 11},
    {"id": 11, "mDebut": 20, "mFin": 40, "hDebut": 11, "hFin": 11},
    {"id": 12, "mDebut": 40, "mFin": 0, "hDebut": 12, "hFin": 12},
    {"id": 13, "mDebut": 0, "mFin": 20, "hDebut": 14, "hFin": 14},
    {"id": 14, "mDebut": 20, "mFin": 40, "hDebut": 14, "hFin": 14},
    {"id": 15, "mDebut": 40, "mFin": 0, "hDebut": 15, "hFin": 15},
    {"id": 16, "mDebut": 0, "mFin": 20, "hDebut": 15, "hFin": 15},
    {"id": 17, "mDebut": 20, "mFin": 40, "hDebut": 15, "hFin": 15},
    {"id": 18, "mDebut": 40, "mFin": 0, "hDebut": 16, "hFin": 16},
    {"id": 19, "mDebut": 0, "mFin": 20, "hDebut": 16, "hFin": 16},
    {"id": 20, "mDebut": 20, "mFin": 40, "hDebut": 16, "hFin": 16},
    {"id": 21, "mDebut": 40, "mFin": 0, "hDebut": 17, "hFin": 17},
    {"id": 22, "mDebut": 0, "mFin": 20, "hDebut": 17, "hFin": 17},
    {"id": 23, "mDebut": 20, "mFin": 40, "hDebut": 17, "hFin": 17},
    {"id": 24, "mDebut": 40, "mFin": 0, "hDebut": 17, "hFin": 17}
  ]
}

```

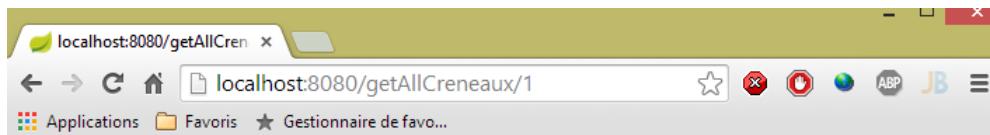
ou bien ceux-ci si le créneau n'existe pas :



The screenshot shows a browser window with the URL `localhost:8080/getAllCreneaux/123`. The page content displays a JSON object indicating an error.

```
{"status": 2, "data": null}
```

ou bien ceux-ci en cas d'erreur d'accès à la base :



2.12.10 L'URL [/getRvMedecinJour/{idMedecin}/{jour}]

L'URL [/getRvMedecinJour/{idMedecin}/{jour}] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```

1. // liste des rendez-vous d'un médecin
2. @RequestMapping(value = "/getRvMedecinJour/{idMedecin}/{jour}", method = RequestMethod.GET)
3. public Reponse getRvMedecinJour(@PathVariable("idMedecin") long idMedecin,
4. @PathVariable("jour") String jour) {
5.     // état de l'application
6.     if (messages != null) {
7.         return new Reponse(-1, messages);
8.     }
9.     // on vérifie la date
10.    Date jourAgenda = null;
11.    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
12.    sdf.setLenient(false);
13.    try {
14.        jourAgenda = sdf.parse(jour);
15.    } catch (ParseException e) {
16.        return new Reponse(3, null);
17.    }
18.    // on récupère le médecin
19.    Reponse réponse = getMedecin(idMedecin);
20.    if (réponse.getStatus() != 0) {
21.        return réponse;
22.    }
23.    Medecin médecin = (Medecin) réponse.getData();
24.    // liste de ses rendez-vous
25.    List<Rv> rvs = null;
26.    try {
27.        rvs = application.getRvMedecinJour(médecin.getId(), jourAgenda);
28.    } catch (Exception e1) {
29.        return new Reponse(4, Static.getErreursForException(e1));
30.    }
31.    // on rend la réponse
32.    return new Reponse(0, Static.getListMapForRvs(rvs));
33. }
```

- ligne 31 : on rend un objet **List<Map<String, Object>>** au lieu d'un objet **List<Rv>**. Rappelons la définition de la classe [Rv] :

```

1. @Entity
2. @Table(name = "rv")
3. public class Rv extends AbstractEntity {
4.     private static final long serialVersionUID = 1L;
5.
6.     // caractéristiques d'un Rv
7.     @Temporal(TemporalType.DATE)
8.     private Date jour;
9.
```

```

10.    // un rv est lié à un client
11.    @ManyToOne(fetch = FetchType.LAZY)
12.    @JoinColumn(name = "id_client")
13.    private Client client;
14.
15.    // un rv est lié à un créneau
16.    @ManyToOne(fetch = FetchType.LAZY)
17.    @JoinColumn(name = "id_creneau")
18.    private Creneau creneau;
19.
20.    // clés étrangères
21.    @Column(name = "id_client", insertable = false, updatable = false)
22.    private long idClient;
23.    @Column(name = "id_creneau", insertable = false, updatable = false)
24.    private long idCreneau;
25.
26. ...
27.
28. }

```

- ligne 11 : le client est recherché avec le mode [FetchType.LAZY] ;
- ligne 18 : le créneau est recherché avec le mode [FetchType.LAZY] ;

Rappelons la requête JPQL qui va chercher les rendez-vous :

```
@Query("select rv from Rv rv left join fetch rv.client c left join fetch rv.creneau cr where cr.medecin.id=?1 and rv.jour=?2")
```

De jointures sont faites explicitement pour ramener les champs [client] et [creneau]. Par ailleurs à cause de la jointure [cr.medecin.id=?1], nous aurons également le médecin. Le médecin va donc apparaître dans la chaîne JSON de chaque rendez-vous. Or cette information dupliquée est en outre inutile. Revenons au code de la méthode :

- ligne 31 : nous construisons nous mêmes le dictionnaire à sérialiser en JSON ;

Le dictionnaire construit pour un rendez-vous est le suivant :

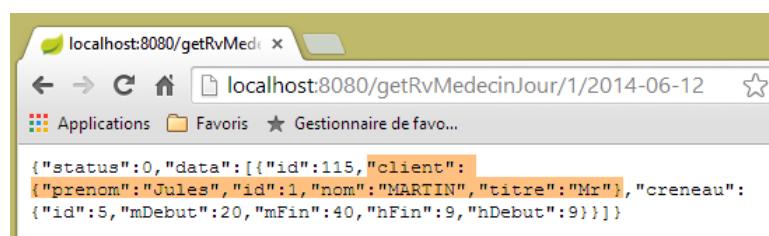
```

1.    // Rv --> Map
2.    public static Map<String, Object> getMapForRv(Rv rv) {
3.        // qq chose à faire ?
4.        if (rv == null) {
5.            return null;
6.        }
7.        // dictionnaire <String, Object>
8.        Map<String, Object> hash = new HashMap<String, Object>();
9.        hash.put("id", rv.getId());
10.       hash.put("client", rv.getClient());
11.       hash.put("creneau", getMapForCreneau(rv.getCreneau()));
12.       // on rend le dictionnaire
13.       return hash;
14.    }

```

- ligne 11 : nous reprenons le dictionnaire de l'objet [Creneau] que nous avons présenté précédemment ;

Les résultats obtenus sont les suivants :



ou encore ceux-ci avec un jour incorrect :



ou encore ceux-ci avec un médecin incorrect :



2.12.11 L'URL [/getAgendaMedecinJour/{idMedecin}/{jour}]

L'URL [/getAgendaMedecinJour/{idMedecin}/{jour}] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```

1. @RequestMapping(value = "/getAgendaMedecinJour/{idMedecin}/{jour}", method = RequestMethod.GET)
2.     public Reponse getAgendaMedecinJour(@PathVariable("idMedecin") long idMedecin, @PathVariable("jour")
3.     String jour) {
4.         // état de l'application
5.         if (messages != null) {
6.             return new Reponse(-1, messages);
7.         }
8.         // on vérifie la date
9.         Date jourAgenda = null;
10.        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
11.        sdf.setLenient(false);
12.        try {
13.            jourAgenda = sdf.parse(jour);
14.        } catch (ParseException e) {
15.            return new Reponse(3, new String[] { String.format("jour [%s] invalide", jour) });
16.        }
17.        // on récupère le médecin
18.        Reponse réponse = getMedecin(idMedecin);
19.        if (réponse.getStatus() != 0) {
20.            return réponse;
21.        }
22.        Medecin médecin = (Medecin) réponse.getData();
23.        // on récupère son agenda
24.        AgendaMedecinJour agenda = null;
25.        try {
26.            agenda = application.getAgendaMedecinJour(médecin.getId(), jourAgenda);
27.        } catch (Exception e1) {
28.            return new Reponse(4, Static.getErreursForException(e1));
29.        }
30.        // ok
31.        return new Reponse(0, Static.getMapForAgendaMedecinJour(agenda));
32.    }

```

- ligne 30, on rend un objet de type `List<Map<String, Object>`.

La méthode [Static.getMapForAgendaMedecinJour] est la suivante :

```

1.     // AgendaMedecinJour --> Map
2.     public static Map<String, Object> getMapForAgendaMedecinJour(AgendaMedecinJour agenda) {
3.         // qq chose à faire ?

```

```

4.     if (agenda == null) {
5.         return null;
6.     }
7.     // dictionnaire <String, Object>
8.     Map<String, Object> hash = new HashMap<String, Object>();
9.     hash.put("medecin", agenda.getMedecin());
10.    hash.put("jour", new SimpleDateFormat("yyyy-MM-dd").format(agenda.getJour()));
11.    List<Map<String, Object>> creneaux = new ArrayList<Map<String, Object>>();
12.    for (CreneauMedecinJour creneau : agenda.getCreneauxMedecinJour()) {
13.        creneaux.add(getMapForCreneauMedecinJour(creneau));
14.    }
15.    hash.put("creneauxMedecin", creneaux);
16.    // on rend le dictionnaire
17.    return hash;
18. }
```

Le dictionnaire construit a trois champs :

- [medecin] : le médecin propriétaire de l'agenda. On a gardé cette information car elle n'est présente qu'une fois alors que dans les cas précédents, elle était répétée dans chaque chaîne JSON ;
- [jour] : le jour de l'agenda ;
- [creneauxMedecin] : la liste des créneaux du médecin avec un éventuel rendez-vous sur ce créneau ;

La méthode [getMapForCreneauMedecinJour] utilisée ligne 13 est la suivante :

```

1.    // CreneauMedecinJour --> map
2.    public static Map<String, Object> getMapForCreneauMedecinJour(CreneauMedecinJour creneau) {
3.        // qq chose à faire ?
4.        if (creneau == null) {
5.            return null;
6.        }
7.        // dictionnaire <String, Object>
8.        Map<String, Object> hash = new HashMap<String, Object>();
9.        hash.put("creneau", getMapForCreneau(creneau.getCreneau()));
10.       hash.put("rv", getMapForRv(creneau.getRv()));
11.       // on rend le dictionnaire
12.       return hash;
13. }
```

- lignes 9-10 : on utilise les dictionnaires déjà étudiés pour les types [Creneau] et [Rv] qui n'embarquent donc pas d'objet [Medecin] ;

Les résultats obtenus sont les suivants :

The screenshot shows a browser window with the URL `localhost:8080/getAgendaMedecinJour/1/2014-06-12`. The page content is a large JSON object representing a doctor's agenda for June 12, 2014. The JSON includes fields for the doctor's name ('Marie'), title ('Mme'), and various appointment slots ('creneauxMedecin') from 08:00 to 18:00.

```
{
  "status": 0,
  "data": {
    "medecin": {
      "prenom": "Marie",
      "id": 1,
      "nom": "PELISSIER",
      "titre": "Mme"
    },
    "creneauxMedecin": [
      {"rv": null, "creneau": {"id": 1, "mDebut": 0, "mFin": 20, "hFin": 8, "hDebut": 8}},
      {"rv": null, "creneau": {"id": 2, "mDebut": 20, "mFin": 40, "hFin": 8, "hDebut": 8}},
      {"rv": null, "creneau": {"id": 3, "mDebut": 40, "mFin": 0, "hFin": 9, "hDebut": 8}},
      {"rv": null, "creneau": {"id": 4, "mDebut": 0, "mFin": 20, "hFin": 9, "hDebut": 9}},
      {"rv": {"id": 115, "client": {"prenom": "Jules", "id": 1, "nom": "MARTIN", "titre": "Mr"}, "creneau": {"id": 5, "mDebut": 20, "mFin": 40, "hFin": 9, "hDebut": 9}}, "creneau": {"id": 6, "mDebut": 40, "mFin": 0, "hFin": 10, "hDebut": 9}},
      {"rv": null, "creneau": {"id": 7, "mDebut": 0, "mFin": 20, "hFin": 10, "hDebut": 10}},
      {"rv": null, "creneau": {"id": 8, "mDebut": 20, "mFin": 40, "hFin": 10, "hDebut": 10}},
      {"rv": null, "creneau": {"id": 9, "mDebut": 40, "mFin": 0, "hFin": 11, "hDebut": 10}},
      {"rv": null, "creneau": {"id": 10, "mDebut": 0, "mFin": 20, "hFin": 11, "hDebut": 11}},
      {"rv": null, "creneau": {"id": 11, "mDebut": 20, "mFin": 40, "hFin": 11, "hDebut": 11}},
      {"rv": null, "creneau": {"id": 12, "mDebut": 40, "mFin": 0, "hFin": 12, "hDebut": 11}},
      {"rv": null, "creneau": {"id": 13, "mDebut": 0, "mFin": 20, "hFin": 14, "hDebut": 11}},
      {"rv": null, "creneau": {"id": 14, "mDebut": 20, "mFin": 40, "hFin": 14, "hDebut": 14}},
      {"rv": null, "creneau": {"id": 15, "mDebut": 40, "mFin": 0, "hFin": 15, "hDebut": 14}},
      {"rv": null, "creneau": {"id": 16, "mDebut": 0, "mFin": 20, "hFin": 15, "hDebut": 15}},
      {"rv": null, "creneau": {"id": 17, "mDebut": 20, "mFin": 40, "hFin": 15, "hDebut": 15}},
      {"rv": null, "creneau": {"id": 18, "mDebut": 40, "mFin": 0, "hFin": 16, "hDebut": 15}},
      {"rv": null, "creneau": {"id": 19, "mDebut": 0, "mFin": 20, "hFin": 16, "hDebut": 16}},
      {"rv": null, "creneau": {"id": 20, "mDebut": 20, "mFin": 40, "hFin": 16, "hDebut": 16}},
      {"rv": null, "creneau": {"id": 21, "mDebut": 40, "mFin": 0, "hFin": 17, "hDebut": 16}},
      {"rv": null, "creneau": {"id": 22, "mDebut": 0, "mFin": 20, "hFin": 17, "hDebut": 17}},
      {"rv": null, "creneau": {"id": 23, "mDebut": 20, "mFin": 40, "hFin": 17, "hDebut": 17}},
      {"rv": null, "creneau": {"id": 24, "mDebut": 40, "mFin": 0, "hFin": 18, "hDebut": 17}}
    ],
    "jour": "2014-06-12"
  }
}
```

ou bien ceux-ci si le jour est erroné :

The screenshot shows a browser window with the URL `localhost:8080/getAgendaMedecinJour/1/2014-06-42`. The response status is 3, indicating an error, and the message is "jour [2014-06-42] invalide".

```
{"status": 3, "data": ["jour [2014-06-42] invalide"]}
```

ou bien ceux-ci si le n° du médecin est invalide :

The screenshot shows a browser window with the URL `localhost:8080/getAgendaMedecinJour/100/2014-06-12`. The response status is 2, and the data is null.

```
{"status": 2, "data": null}
```

2.12.12 L'URL [/getMedecinById/{id}]

L'URL [/getMedecinById/{id}] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```

1.   @RequestMapping(value = "/getMedecinById/{id}", method = RequestMethod.GET)
2.   public Reponse getMedecinById(@PathVariable("id") long id) {
3.     // état de l'application
4.     if (messages != null) {
5.       return new Reponse(-1, messages);
6.     }
7.     // on récupère le médecin
8.     return getMedecin(id);

```

9. }

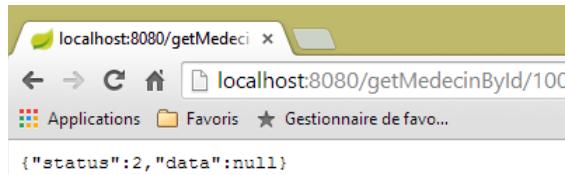
Ligne 8, la méthode [getMedecin] est la suivante :

```
1.     private Reponse getMedecin(long id) {
2.         // on récupère le médecin
3.         Medecin médecin = null;
4.         try {
5.             médecin = application.getMedecinById(id);
6.         } catch (Exception e1) {
7.             return new Reponse(1, Static.getErreursForException(e1));
8.         }
9.         // médecin existant ?
10.        if (métècin == null) {
11.            return new Reponse(2, null);
12.        }
13.        // ok
14.        return new Reponse(0, médecin);
15.    }
```

Les résultats obtenus sont les suivants :



ou bien ceux-ci si le n° du médecin est incorrect :



2.12.13 L'URL [/getClientById/{id}]

L'URL [/getClientById/{id}] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```
1.     @RequestMapping(value = "/getClientById/{id}", method = RequestMethod.GET)
2.     public Reponse getClientById(@PathVariable("id") long id) {
3.         // état de l'application
4.         if (messages != null) {
5.             return new Reponse(-1, messages);
6.         }
7.         // on récupère le client
8.         return getClient(id);
9.     }
```

Ligne 8, la méthode [getClient] est la suivante :

```
1.     private Reponse getClient(long id) {
2.         // on récupère le client
3.         Client client = null;
4.         try {
5.             client = application.getClientById(id);
6.         } catch (Exception e1) {
7.             return new Reponse(1, Static.getErreursForException(e1));
```

```

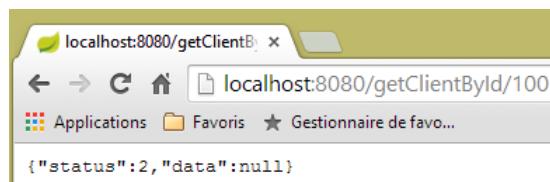
8.      }
9.      // client existant ?
10.     if (client == null) {
11.         return new Reponse(2, null);
12.     }
13.     // ok
14.     return new Reponse(0, client);
15. }

```

Les résultats obtenus sont les suivants :



ou bien ceux-ci si le n° du client est incorrect :



2.12.14 L'URL [/getCreneauById/{id}]

L'URL [/getCreneauById/{id}] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```

1.  @RequestMapping(value = "/getCreneauById/{id}", method = RequestMethod.GET)
2.  public Reponse getCreneauById(@PathVariable("id") long id) {
3.      // état de l'application
4.      if (messages != null) {
5.          return new Reponse(-1, messages);
6.      }
7.      // on récupère le créneau
8.      Reponse réponse = getCreneau(id);
9.      if (réponse.getStatus() == 0) {
10.          réponse.setData(Static.getMapForCreneau((Creneau) réponse.getData()));
11.      }
12.      // résultat
13.      return réponse;
14. }

```

Ligne 8, la méthode [getCreneau] est la suivante :

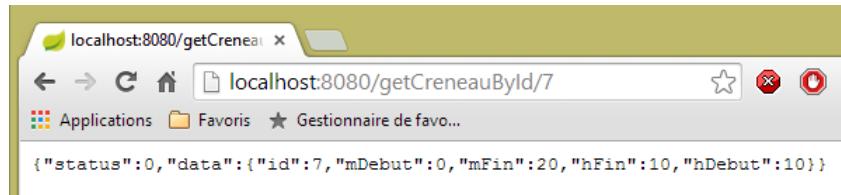
```

1.  private Reponse getCreneau(long id) {
2.      // on récupère le créneau
3.      Creneau créneau = null;
4.      try {
5.          créneau = application.getCreneauById(id);
6.      } catch (Exception e1) {
7.          return new Reponse(1, Static.getErreursForException(e1));
8.      }
9.      // créneau existant ?
10.     if (créneau == null) {
11.         return new Reponse(2, null);
12.     }
13.     // ok
14.     return new Reponse(0, créneau);

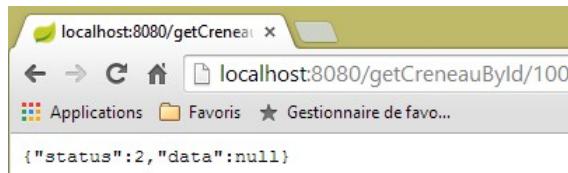
```

15. }

Les résultats obtenus sont les suivants :



ou ceux-ci si le n° du créneau est incorrect :



2.12.15 L'URL [/getRvById/{id}]

L'URL [/getRvById/{id}] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```
1.  @RequestMapping(value = "/getRvById/{id}", method = RequestMethod.GET)
2.  public Reponse getRvById(@PathVariable("id") long id) {
3.      // état de l'application
4.      if (messages != null) {
5.          return new Reponse(-1, messages);
6.      }
7.      // on récupère le rv
8.      Reponse réponse = getRv(id);
9.      if (réponse.getStatus() == 0) {
10.         réponse.setData(Static.getMapForRv2((Rv) réponse.getData()));
11.     }
12.    // résultat
13.    return réponse;
14. }
```

Ligne 8, la méthode [getRv] est la suivante :

```
1.  private Reponse getRv(long id) {
2.      // on récupère le Rv
3.      Rv rv = null;
4.      try {
5.          rv = application.getRvById(id);
6.      } catch (Exception e1) {
7.          return new Reponse(1, Static.getErreursForException(e1));
8.      }
9.      // Rv existant ?
10.     if (rv == null) {
11.         return new Reponse(2, null);
12.     }
13.     // ok
14.     return new Reponse(0, rv);
15. }
```

Ligne 10, la méthode [Static.getMapForRv2] est la suivante :

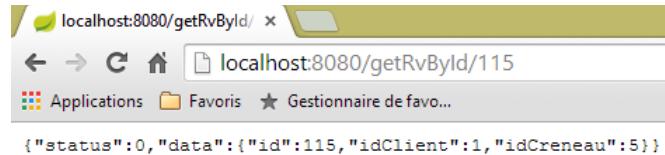
```
1.  // Rv --> Map
2.  public static Map<String, Object> getMapForRv2(Rv rv) {
3.      // qq chose à faire ?
```

```

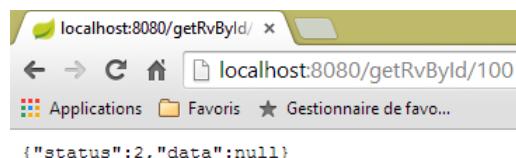
4.     if (rv == null) {
5.         return null;
6.     }
7.     // dictionnaire <String, Object>
8.     Map<String, Object> hash = new HashMap<String, Object>();
9.     hash.put("id", rv.getId());
10.    hash.put("idClient", rv.getIdClient());
11.    hash.put("idCreneau", rv.getIdCreneau());
12.    // on rend le dictionnaire
13.    return hash;
14. }

```

Les résultats obtenus sont les suivants :



ou bien ceux-ci si le n° du rendez-vous est incorrect :



2.12.16 L'URL [/ajouterRv]

L'URL [/ajouterRv] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```

1. @RequestMapping(value = "/ajouterRv", method = RequestMethod.POST, consumes = "application/json;
   charset=UTF-8")
2.     public Reponse ajouterRv(@RequestBody PostAjouterRv post) {
3.         // état de l'application
4.         if (messages != null) {
5.             return new Reponse(-1, messages);
6.         }
7.         // on récupère les valeurs postées
8.         String jour = post.getJour();
9.         long idCreneau = post.getIdCreneau();
10.        long idClient = post.getIdClient();
11.        // on vérifie la date
12.        Date jourAgenda = null;
13.        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
14.        sdf.setLenient(false);
15.        try {
16.            jourAgenda = sdf.parse(jour);
17.        } catch (ParseException e) {
18.            return new Reponse(6, null);
19.        }
20.        // on récupère le créneau
21.        Reponse réponse = getCreneau(idCreneau);
22.        if (réponse.getStatus() != 0) {
23.            return réponse;
24.        }
25.        Creneau créneau = (Creneau) réponse.getData();
26.        // on récupère le client
27.        réponse = getClient(idClient);
28.        if (réponse.getStatus() != 0) {
29.            réponse.incrStatusBy(2);

```

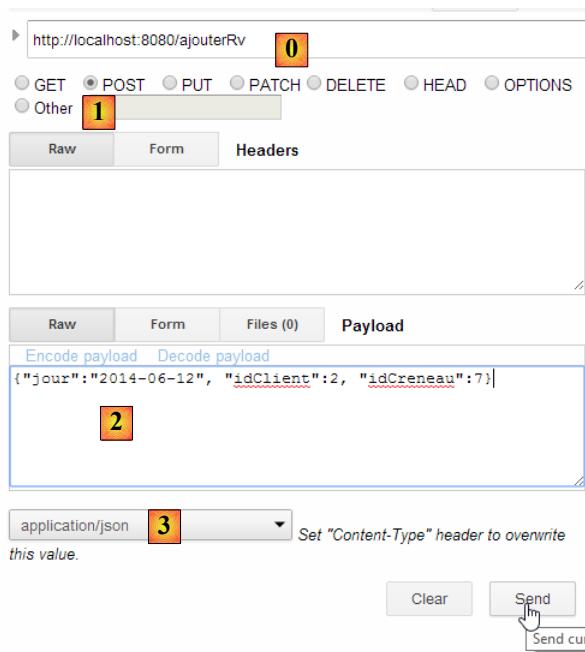
```

30.         return réponse;
31.     }
32.     Client client = (Client) réponse.getData();
33.     // on ajoute le Rv
34.     Rv rv = null;
35.     try {
36.         rv = application.ajouterRv(jourAgenda, créneau, client);
37.     } catch (Exception e1) {
38.         return new Reponse(5, Static.getErreursForException(e1));
39.     }
40.     // on rend la réponse
41.     return new Reponse(0, Static.getMapForRv(rv));
42. }

```

Il n'y a là rien qui n'ait été déjà vu. Ligne 41, on rend le rendez-vous qui a été ajouté ligne 36.

Les résultats obtenus ressemblent à ceci avec le client [Advanced Rest Client] :



Status **200 OK** ⚡ Loading time: 136 ms

Request headers

- User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36
- Origin: chrome-extension://fhgmllofdffdnphfgcellkdfbfobjelo
- Content-Type: application/json** 4
- Accept: */*
- Accept-Encoding: gzip, deflate, sdch
- Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

Response headers

- Server: Apache-Coyote/1.1
- Content-Type: application/json; charset=UTF-8** 5
- Transfer-Encoding: chunked
- Date: Thu, 12 Jun 2014 09:28:51 GMT

Raw JSON Response

```

Copy to clipboard Save as file
{
    status: 0
    -data: {
        id: 117 6
        -client: {
            prenom: "Christine"
            id: 2
            nom: "GERMAN"
            titre: "Mme"
        }
        -creneau: {
            id: 7
            mDebut: 0
            mFin: 20
            hFin: 10
            hDebut: 10
        }
    }
}

```

ou bien à ceci si par exemple on donne un n° de créneau inexistant :

http://localhost:8080/ajouterRv

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Headers

Raw Form Files (0) Payload

Encode payload Decode payload

```
{"jour": "2014-06-12", "idClient": 1, "idCreneau": 100}
```

application/json Set "Content-Type" header to overwrite this value.

Status **200 OK** ⚡ Loading time: 276 ms

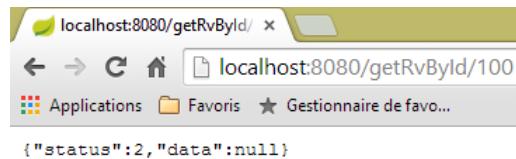
Request headers

```
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2049.0 Safari/537.36
Origin: chrome-extension://hgmloofddffdi
Content-Type: application/json
Accept: */*
Accept-Encoding: gzip, deflate, sdch
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6
```

Response headers

```
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 12 Jun 2014 15:10:26 GMT
```

Raw	JSON	Response
Copy to clipboard	Save as file	
<pre>{ status: 2 data: null }</pre>		



2.12.17 L'URL [/supprimerRv]

L'URL [/supprimerRv] est traitée par la méthode suivante du contrôleur [RdvMedecinsController] :

```

1. @RequestMapping(value = "/supprimerRv", method = RequestMethod.POST, consumes = "application/json;
   charset=UTF-8")
2. public Reponse supprimerRv(@RequestBody PostSupprimerRv post) {
3.     // état de l'application
4.     if (messages != null) {
5.         return new Reponse(-1, messages);
6.     }
7.     // on récupère les valeurs postées
8.     long idRv = post.getIdRv();
9.     // on récupère le rv
10.    Reponse réponse = getRv(idRv);
11.    if (réponse.getStatus() != 0) {
12.        return réponse;
13.    }
14.    // suppression du rv
15.    try {
16.        application.supprimerRv(idRv);
17.    } catch (Exception e1) {
18.        return new Reponse(3, Static.getErreursForException(e1));
19.    }
20.    // ok
21.    return new Reponse(0, null);
22. }
```

Les résultats obtenus sont les suivants :

http://localhost:8080/supprimerRv **1**

GET POST PUT PATCH DELETE HEADERS

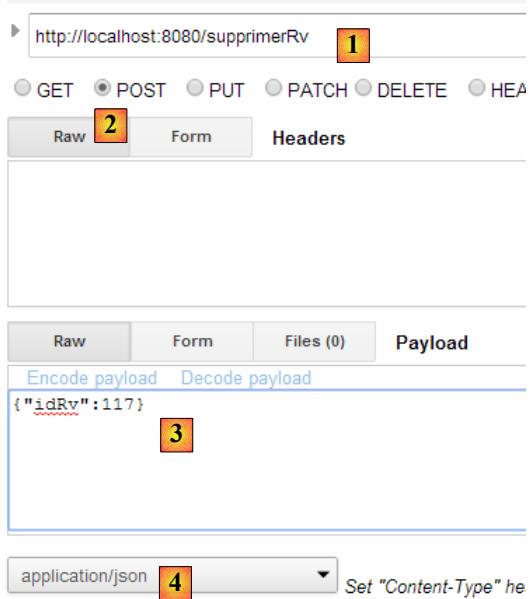
Raw **2** Form Headers

Raw Form Files (0) Payload

Encode payload Decode payload

```
{"idRv":117} 3
```

application/json **4** Set "Content-Type" header



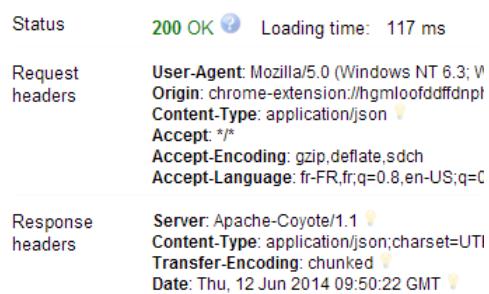
Status **200 OK** Loading time: 117 ms

Request headers

```
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2049.0 Safari/537.36
Origin: chrome-extension://hgmloofddfdnpotckmafjndnmejjgdld
Content-Type: application/json
Accept: */*
Accept-Encoding: gzip,deflate,sdch
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.5
```

Response headers

```
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 12 Jun 2014 09:50:22 GMT
```



Raw JSON Response

Copy to clipboard Save as file

```
{
  "status": 0
  "data": null 5
}
```



ou bien ceux-ci si le n° du rendez-vous n'existe pas :

Status	200 OK	Loading time: 18 ms
Request headers	User-Agent: Mozilla/5.0 (Windows NT 6.3 Origin: chrome-extension://hgmlloofddffdr Content-Type: application/json Accept: */* Accept-Encoding: gzip, deflate, sdch Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.5	
Response headers	Server: Apache-Coyote/1.1 Content-Type: application/json;charset=UTF-8 Transfer-Encoding: chunked Date: Thu, 12 Jun 2014 15:21:34 GMT	

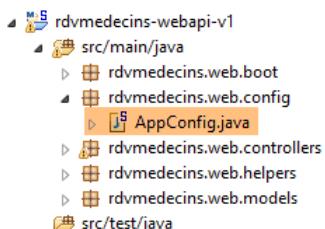
Raw **JSON** **Response**

[Copy to clipboard](#) [Save as file](#)

```
{
  "status": 2
  "data": null
}
```

Nous en avons terminé avec le contrôleur. Nous voyons maintenant comment configurer le projet.

2.12.18 Configuration du service web



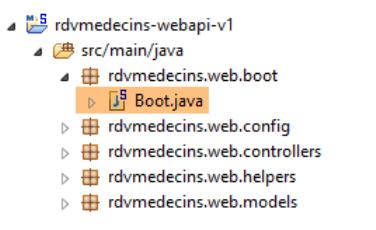
La classe de configuration [AppConfig] est la suivante :

```

1. package rdvmedecins.web.config;
2.
3. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Import;
6.
7. import rdvmedecins.config.DomainAndPersistenceConfig;
8.
9. @EnableAutoConfiguration
10. @ComponentScan(basePackages = { "rdvmedecins.web" })
11. @Import({ DomainAndPersistenceConfig.class })
12. public class AppConfig {
13.
14. }
```

- ligne 9 : on se met en mode [AutoConfiguration] afin que Spring Boot puisse configurer le projet en fonction des archives qu'il trouvera dans le Classpath du projet ;
- ligne 10 : on demande à ce que les composants Spring soient cherchés dans le package [rdvmedecins.web] et ses descendants. C'est ainsi que seront découverts les composants :
 - [@RestController RdvMedecinsController] dans le package [rdvmedecins.web.controllers] ;
 - [@Component ApplicationModel] dans le package [rdvmedecins.web.models] ;
- ligne 11 : on importe la classe [DomainAndPersistenceConfig] qui configure le projet [rdvmedecins-metier-dao] afin d'avoir accès aux beans de ce projet ;

2.12.19 La classe exécutable du service web



La classe [Boot] est la suivante :

```
1. package rdvmedecins.web.boot;
2.
3. import org.springframework.boot.SpringApplication;
4.
5. import rdvmedecins.web.config.AppConfig;
6.
7. public class Boot {
8.
9.     public static void main(String[] args) {
10.         SpringApplication.run(AppConfig.class, args);
11.     }
12. }
```

Ligne 10, la méthode statique [SpringApplication.run] est exécutée avec comme premier paramètre, la classe [AppConfig] de configuration du projet. Cette méthode va procéder à l'auto-configuration du projet, lancer le serveur Tomcat embarqué dans les dépendances et y déployer le contrôleur [RdvMedecinsController].

Les logs à l'exécution sont les suivants :

```
1. .
2. /\_\_/\_ .\_\_-\_(_)_\_--\_\_\\ \_
3. ( ( )\_\_| .|_| .|_| .\_\_`_| \_\_\\ )
4. \_\_/\_\_)| [D]| | | | | | (|_| )) ) )
5. ' |_\_ | .|_| | |_\_ |_\_ , | / / / /
6. ======|_|=====|_|=/_/_/_/
7. :: Spring Boot ::      (v1.0.0.RELEASE)
8.
9. 2014-06-12 17:30:41.261  INFO 9388 --- [           main] rdvmmedecins.web.boot.Boot          :
   Starting Boot on Gportpers3 with PID 9388 (D:\data\istia-1314\polys\istia\angularjs-
   spring4\dvp\rdvmmedecins-webapi\target\classes started by ST)
10. 2014-06-12 17:30:41.306  INFO 9388 --- [           main] ationConfigEmbeddedWebApplicationContext :
   Refreshing
   org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@ale932e: startup
   date [Thu Jun 12 17:30:41 CEST 2014]; root of context hierarchy
11. 2014-06-12 17:30:42.058  INFO 9388 --- [           main] o.s.b.f.s.DefaultListableBeanFactory      :
   Overriding bean definition for bean 'org.springframework.boot.autoconfigure.AutoConfigurationPackages':
   replacing [Generic bean: class
   [org.springframework.boot.autoconfigure.AutoConfigurationPackages$BasePackages]; scope=; abstract=false;
   lazyInit=false; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false;
   factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null] with [Generic
   bean: class [org.springframework.boot.autoconfigure.AutoConfigurationPackages$BasePackages]; scope=;
   abstract=false; lazyInit=false; autowireMode=0; dependencyCheck=0; autowireCandidate=true;
   primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null;
   destroyMethodName=null]
12. 2014-06-12 17:30:42.866  INFO 9388 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean
   'org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration' of type [class
   org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration$
   $EnhancerBySpringCGLIB$$fd7a7b18] is not eligible for getting processed by all BeanPostProcessors (for
   example: not eligible for auto-proxying)
13. 2014-06-12 17:30:42.900  INFO 9388 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean
   'transactionAttributeSource' of type [class
   org.springframework.transaction.annotation.AnnotationTransactionAttributeSource] is not eligible for
   getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
14. 2014-06-12 17:30:42.915  INFO 9388 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean
   'transactionInterceptor' of type [class
```

org.springframework.transaction.interceptor.TransactionInterceptor] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)

15. 2014-06-12 17:30:42.920 INFO 9388 --- [main] trationDelegate\$BeanPostProcessorChecker : Bean 'org.springframework.transaction.config.internalTransactionAdvisor' of type [class org.springframework.transaction.interceptor.BeanFactoryTransactionAttributeSourceAdvisor] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)

16. 2014-06-12 17:30:43.164 INFO 9388 --- [main] .t.TomcatEmbeddedServletContainerFactory : Server initialized with port: 8080

17. 2014-06-12 17:30:43.403 INFO 9388 --- [main] o.apache.catalina.core.StandardService : Starting service Tomcat

18. 2014-06-12 17:30:43.403 INFO 9388 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/7.0.52

19. 2014-06-12 17:30:43.582 INFO 9388 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

20. 2014-06-12 17:30:43.582 INFO 9388 --- [ost-startStop-1] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2279 ms

21. 2014-06-12 17:30:44.117 INFO 9388 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean : Mapping servlet: 'dispatcherServlet' to [/*]

22. 2014-06-12 17:30:44.119 INFO 9388 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]

23. 2014-06-12 17:30:44.662 INFO 9388 --- [main] j.LocalContainerEntityManagerFactoryBean : Building JPA container EntityManagerFactory for persistence unit 'default'

24. 2014-06-12 17:30:44.707 INFO 9388 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [

25. name: default

26. ...]

27. 2014-06-12 17:30:44.839 INFO 9388 --- [main] org.hibernate.Version : HHH000412: Hibernate Core {4.3.1.Final}

28. 2014-06-12 17:30:44.842 INFO 9388 --- [main] org.hibernate.cfg.Environment : HHH000206: hibernate.properties not found

29. 2014-06-12 17:30:44.844 INFO 9388 --- [main] org.hibernate.cfg.Environment : HHH000021: Bytecode provider name : javassist

30. 2014-06-12 17:30:45.189 INFO 9388 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {4.0.4.Final}

31. 2014-06-12 17:30:45.616 INFO 9388 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect

32. 2014-06-12 17:30:45.783 INFO 9388 --- [main] o.h.h.i.ast.ASTQueryTranslatorFactory : HHH000397: Using ASTQueryTranslatorFactory

33. 2014-06-12 17:30:46.729 INFO 9388 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

34. 2014-06-12 17:30:46.825 INFO 9388 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/getRvMedecinJour/{idMedecin}/ {jour}}],methods=[GET],params=[],headers=[],consumes=[],produces[],custom[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.getRvMedecinJour(long,java.lang.String)

35. 2014-06-12 17:30:46.826 INFO 9388 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/getAllCreneaux/ {idMedecin}}],methods=[GET],params=[],headers[],consumes[],produces[],custom[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.getAllCreneaux(long)

36. 2014-06-12 17:30:46.826 INFO 9388 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/getAgendaMedecinJour/{idMedecin}/ {jour}}],methods=[GET],params[],headers[],consumes[],produces[],custom[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.getAgendaMedecinJour(long,java.lang.String)

37. 2014-06-12 17:30:46.826 INFO 9388 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/getAllMedecins},methods=[GET],params[],headers[],consumes[],produces[],custom[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.getAllMedecins()

38. 2014-06-12 17:30:46.826 INFO 9388 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/getMedecinById/{id}}],methods=[GET],params[],headers[],consumes[],produces[],custom[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.getMedecinById(long)

39. 2014-06-12 17:30:46.827 INFO 9388 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/getCreneauById/{id}}],methods=[GET],params[],headers[],consumes[],produces[],custom[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.getCreneauById(long)

40. 2014-06-12 17:30:46.827 INFO 9388 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/getClientById/{id}}],methods=[GET],params[],headers[],consumes[],produces[],custom[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.getClientById(long)

41. 2014-06-12 17:30:46.827 INFO 9388 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/getRvById/{id}}],methods=[GET],params[],headers[],consumes[],produces[],custom[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.getRvById(long)

```

42. 2014-06-12 17:30:46.827 INFO 9388 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/getAllClients},methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.getAllClients()
43. 2014-06-12 17:30:46.827 INFO 9388 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/ajouterRv},methods=[POST],params=[],headers=[],consumes=[application/json;charset=UTF-8],produces=[],custom=[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.ajouterRv(rdvmedecins.web.models.PostAjouterRv)
44. 2014-06-12 17:30:46.828 INFO 9388 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[{/supprimerRv},methods=[POST],params=[],headers=[],consumes=[application/json;charset=UTF-8],produces=[],custom=[]]" onto public rdvmedecins.web.models.Reponse rdvmedecins.web.controllers.RdvMedecinsController.supprimerRv(rdvmedecins.web.models.PostSupprimerRv)
45. 2014-06-12 17:30:46.851 INFO 9388 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
46. 2014-06-12 17:30:46.851 INFO 9388 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
47. 2014-06-12 17:30:47.131 INFO 9388 --- [           main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup
48. 2014-06-12 17:30:47.169 INFO 9388 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080/http
49. 2014-06-12 17:30:47.170 INFO 9388 --- [           main] rdvmedecins.web.boot.Boot : Started Boot in 6.302 seconds (JVM running for 6.906)
50. 2014-06-12 17:30:55.520 INFO 9388 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring FrameworkServlet 'dispatcherServlet'
51. 2014-06-12 17:30:55.520 INFO 9388 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet 'dispatcherServlet': initialization started
52. 2014-06-12 17:30:55.538 INFO 9388 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet 'dispatcherServlet': initialization completed in 18 ms

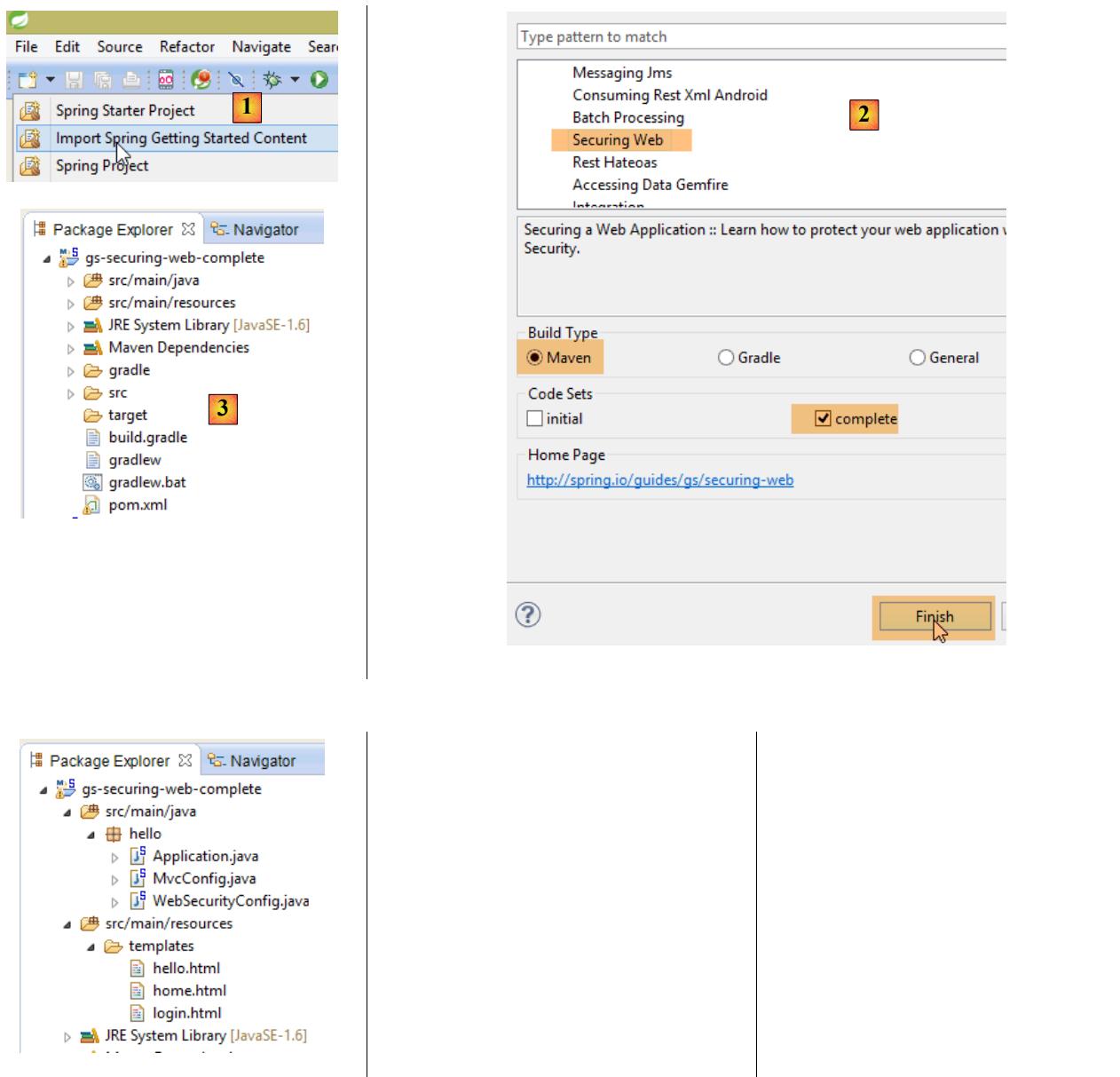
```

- ligne 17 : le serveur Tomcat démarre ;
- lignes 23-31 : les couches [métier, DAO, JPA] s'initialisent ;
- ligne 34 : la méthode traitant l'URL [/getRvMedecinJour/{idMedecin}/{jour}] a été découverte. Ce processus de découverte des méthodes du contrôleur se répète jusqu'à la ligne 44 ;
- ligne 52 : la servlet de Spring MVC [DispatcherServlet] est prête à répondre aux demandes de clients web ;

Nous avons désormais un service web opérationnel interrogeable avec un client web. Nous abordons maintenant la sécurisation de ce service : nous voulons que seules certaines personnes puissent gérer les rendez-vous des médecins. Nous allons utiliser pour cela le framework Spring Security, une branche de l'écosystème Spring.

2.13 Introduction à Spring Security

Nous allons de nouveau importer un guide Spring en suivant les étapes 1 à 3 ci-dessous :



Le projet se compose des éléments suivants :

- dans le dossier [templates], on trouve les pages HTML du projet ;
- [Application] : est la classe exécutable du projet ;
- [MvcConfig] : est la classe de configuration de Spring MVC ;
- [WebSecurityConfig] : est la classe de configuration de Spring Security ;

2.13.1 Configuration Maven

Le projet [3] est un projet Maven. Examinons son fichier [pom.xml] pour connaître ses dépendances :

```

1.   <parent>
2.     <groupId>org.springframework.boot</groupId>
3.     <artifactId>spring-boot-starter-parent</artifactId>
4.     <version>1.1.1.RELEASE</version>
5.   </parent>
6.
7.   <dependencies>
8.     <dependency>
9.       <groupId>org.springframework.boot</groupId>
10.      <artifactId>spring-boot-starter-thymeleaf</artifactId>
11.    </dependency>
```

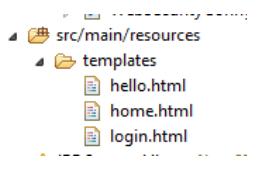
```

12.      <dependency>
13.          <groupId>org.springframework.boot</groupId>
14.          <artifactId>spring-boot-starter-security</artifactId>
15.      </dependency>
16.  </dependencies>

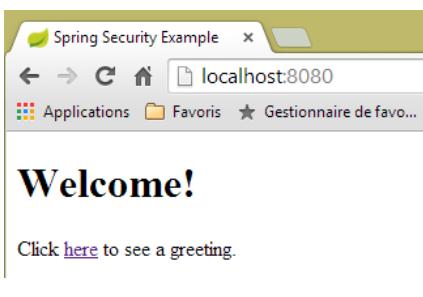
```

- lignes 1-5 : le projet est un projet Spring Boot ;
- lignes 8-11 : dépendance sur le framework [Thymeleaf] qui permet de construire des pages HTML dynamiques. Ce framework peut remplacer les pages JSP (Java Server Pages) qui jusqu'à un passé récent étaient par défaut, le framework de vues de Spring MVC ;
- lignes 12-15 : dépendance sur le framework Spring Security ;

2.13.2 Les vues Thymeleaf



La vue [home.html] est la suivante :



```

1.  <!DOCTYPE html>
2.  <html xmlns="http://www.w3.org/1999/xhtml"
3.      xmlns:th="http://www.thymeleaf.org"
4.      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
5.  <head>
6.  <title>Spring Security Example</title>
7.  </head>
8.  <body>
9.      <h1>Welcome!</h1>
10.
11.     <p>
12.         Click <a th:href="@{/hello}">here</a> to see a greeting.
13.     </p>
14.  </body>
15. </html>

```

- les attributs [**th:xx**] sont des attributs Thymeleaf. Ils sont interprétés par Thymeleaf avant que la page HTML ne soit envoyée au client. Celui ne les voit pas ;
- ligne 12 : l'attribut [**th:href="@{/hello}"**] va générer l'attribut [href] de la balise <a>. La valeur [**@{/hello}**] va générer le chemin [<context>/hello] où [context] est le contexte de l'application web ;

Le code HTML généré est le suivant :

```

1.  <!DOCTYPE html>
2.
3.  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-
   springsecurity3">
4.  <head>
5.  <title>Spring Security Example</title>

```

```

6. </head>
7. <body>
8.   <h1>Welcome!</h1>
9.   <p>
10.     Click <a href="/hello">here</a> to see a greeting.
11.   </p>
12. </body>
13. </html>

```

- ligne 10 : le contexte de l'application est la racine / ;

La vue [hello.html] est la suivante :



```

1.  <!DOCTYPE html>
2.  <html xmlns="http://www.w3.org/1999/xhtml"
3.    xmlns:th="http://www.thymeleaf.org"
4.    xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
5.  <head>
6.  <title>Hello World!</title>
7.  </head>
8.  <body>
9.    <h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
10.   <form th:action="@{/Logout}" method="post">
11.     <input type="submit" value="Sign Out" />
12.   </form>
13. </body>
14. </html>

```

- ligne 9 : L'attribut [th:inline="text"] va générer le texte de la balise <h1>. Ce texte contient une expression \${ qui doit être évaluée. L'élément [[\${#httpServletRequest.remoteUser}]] est la valeur de l'attribut [RemoteUser] de la requête HTTP courante. C'est le nom de l'utilisateur connecté ;
- ligne 10 : un formulaire HTML. L'attribut [th:action="@{/Logout}"] va générer l'attribut [action] de la balise [form]. La valeur [@{/Logout}] va générer le chemin [<context>/logout] où [context] est le contexte de l'application web ;

Le code HTML généré est le suivant :

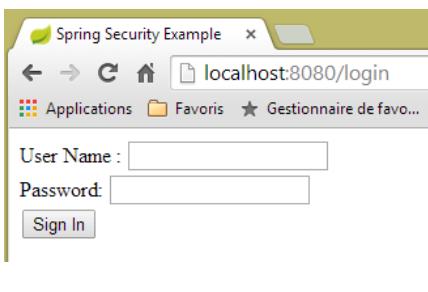
```

1.  <!DOCTYPE html>
2.
3.  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-
   springsecurity3">
4.  <head>
5.  <title>Hello World!</title>
6.  </head>
7.  <body>
8.    <h1>Hello user!</h1>
9.    <form method="post" action="/logout">
10.      <input type="submit" value="Sign Out" />
11.      <input type="hidden" name="_csrf" value="c60cf557-1f3b-415f-a628-39380de7b69a" />
12.    </body>
13.  </html>

```

- ligne 8 : la traduction de Hello [[\${#httpServletRequest.remoteUser}]];
- ligne 9 : la traduction de @{/Logout} ;
- ligne 11 : un champ caché appelé (attribut name) _csrf ;

La dernière vue [login.html] est la suivante :



```
1.  <!DOCTYPE html>
2.  <html xmlns="http://www.w3.org/1999/xhtml"
3.      xmlns:th="http://www.thymeleaf.org"
4.      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
5.  <head>
6.  <title>Spring Security Example</title>
7.  </head>
8.  <body>
9.      <div th:if="${param.error}">Invalid username and password.</div>
10.     <div th:if="${param.logout}">You have been logged out.</div>
11.     <form th:action="@{/Login}" method="post">
12.         <div>
13.             <label> User Name : <input type="text" name="username" />
14.             </label>
15.         </div>
16.         <div>
17.             <label> Password: <input type="password" name="password" />
18.             </label>
19.         </div>
20.         <div>
21.             <input type="submit" value="Sign In" />
22.         </div>
23.     </form>
24. </body>
25. </html>
```

- ligne 9 : l'attribut [th:if="\${param.error}"] fait que la balise <div> ne sera générée que si l'URL qui affiche la page de login contient le paramètre [error] (<http://context/login?error>);
- ligne 10 : l'attribut [th:if="\${param.logout}"] fait que la balise <div> ne sera générée que si l'URL qui affiche la page de login contient le paramètre [logout] (<http://context/login?logout>);
- lignes 11-23 : un formulaire HTML ;
- ligne 11 : le formulaire sera posté à l'URL [<context>/login] où <context> est le contexte de l'application web ;
- ligne 13 : un champ de saisie nommé [username] ;
- ligne 17 : un champ de saisie nommé [password] ;

Le code HTML généré est le suivant :

```
1.  <!DOCTYPE html>
2.
3.  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-
4.      springsecurity3">
5.  <head>
6.  <title>Spring Security Example</title>
7.  </head>
8.
9.      <form method="post" action="/login">
10.         <div>
11.             <label> User Name : <input type="text" name="username" />
12.             </label>
13.         </div>
14.         <div>
15.             <label> Password: <input type="password" name="password" />
16.             </label>
```

```

17.      </div>
18.      <div>
19.          <input type="submit" value="Sign In" />
20.      </div>
21.      <input type="hidden" name="_csrf" value="c60cf557-1f3b-415f-a628-39380de7b69a" /></form>
22.  </body>
23. </html>

```

On notera ligne 21 que Thymeleaf a ajouté un champ caché nommé [_csrf].

2.13.3 Configuration Spring MVC



La classe [MvcConfig] configure le framework Spring MVC :

```

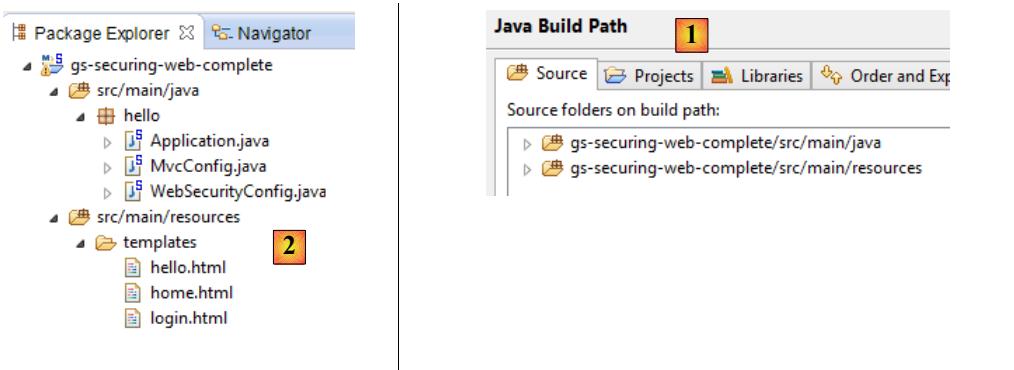
1. package hello;
2.
3. import org.springframework.context.annotation.Configuration;
4. import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
5. import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
6.
7. @Configuration
8. public class MvcConfig extends WebMvcConfigurerAdapter {
9.
10.    @Override
11.    public void addViewControllers(ViewControllerRegistry registry) {
12.        registry.addViewController("/home").setViewName("home");
13.        registry.addViewController("/").setViewName("home");
14.        registry.addViewController("/hello").setViewName("hello");
15.        registry.addViewController("/login").setViewName("login");
16.    }
17.
18. }

```

- ligne 7 : l'annotation [@Configuration] fait de la classe [MvcConfig] une classe de configuration ;
- ligne 8 : la classe [MvcConfig] étend la classe [WebMvcConfigurerAdapter] pour en redéfinir certaines méthodes ;
- ligne 10 : redéfinition d'une méthode de la classe parent ;
- lignes 11-16 : la méthode [addViewControllers] permet d'associer des URL à des vues HTML. Les associations suivantes y sont faites :

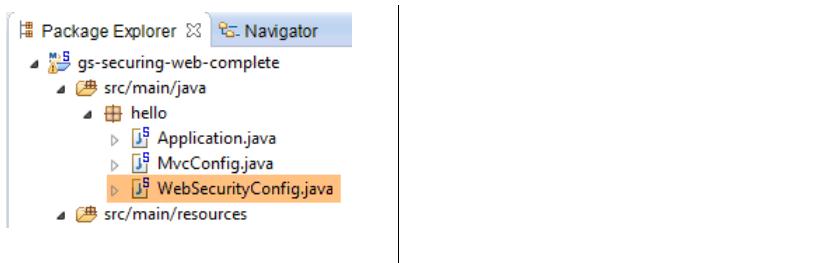
URL	vue
/, /home	/templates/home.html
/hello	/templates/hello.html
/login	/templates/login.html

Le suffixe [html] et le dossier [templates] sont les valeurs par défaut utilisées par Thymeleaf. Elles peuvent être changées par configuration. Le dossier [templates] doit être à la racine du Classpath du projet :



Ci-dessus [1], les dossiers [main] et [resources] sont tous les deux des dossier source (source folders). Cela implique que leur contenu sera à la racine du Classpath du projet. Donc en [2], les dossiers [hello] et [templates] seront à la racine du Classpath.

2.13.4 Configuration Spring Security



La classe [WebSecurityConfig] configure le framework Spring Security :

```

1. package hello;
2.
3. import org.springframework.context.annotation.Configuration;
4. import
5.     org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
6. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
7. import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
8. import org.springframework.security.config.annotation.web.servlet.configuration.EnableWebMvcSecurity;
9.
10. @Configuration
11. @EnableWebMvcSecurity
12. public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
13.     @Override
14.     protected void configure(HttpSecurity http) throws Exception {
15.         http.authorizeRequests().antMatchers("/", "/home").permitAll().anyRequest().authenticated();
16.         http.formLogin().loginPage("/login").permitAll().and().logout().permitAll();
17.     }
18.     @Override
19.     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
20.         auth.inMemoryAuthentication().withUser("user").password("password").roles("USER");
21.     }
22. }
```

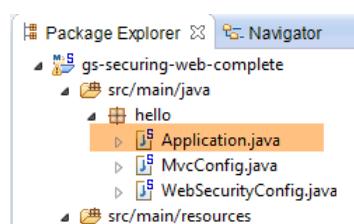
- ligne 9 : l'annotation [@Configuration] fait de la classe [WebSecurityConfig] une classe de configuration ;
- ligne 10 : l'annotation [@EnableWebSecurity] fait de la classe [WebSecurityConfig] une classe de configuration de Spring Security ;
- ligne 11 : la classe [WebSecurity] étend la classe [WebSecurityConfigurerAdapter] pour en redéfinir certaines méthodes ;
- ligne 12 : redéfinition d'une méthode de la classe parent ;
- lignes 13- 16 : la méthode [configure(HttpSecurity http)] est redéfinie pour définir les droits d'accès aux différentes URL de l'application ;
- ligne 14 : la méthode [http.authorizeRequests()] permet d'associer des URL à des droits d'accès. Les associations suivantes y sont faites :

URL	règle	code
-----	-------	------

/, /home	accès sans être authentifié	<code>http.authorizeRequests().antMatchers("/", "/home").permitAll()</code>
autres URL	accès authentifié uniquement	<code>http.anyRequest().authenticated();</code>

- ligne 15 : définit la méthode d'authentification. L'authentification se fait via un formulaire d'URL [/login] accessible à tous [http.formLogin().loginPage("/login").permitAll()]. La déconnexion (logout) est également accessible à tous.
- lignes 19-21 : redéfinissent la méthode [configure(AuthenticationManagerBuilder auth)] qui gère les utilisateurs ;
- ligne 20 : l'autentification se fait avec des utilisateurs définis en "dur" [auth.inMemoryAuthentication()]. Un utilisateur est ici défini avec le login [user], le mot de passe [password] et le rôle [USER]. On peut accorder les mêmes droits à des utilisateurs ayant le même rôle ;

2.13.5 Classe exécutable



La classe [Application] est la suivante :

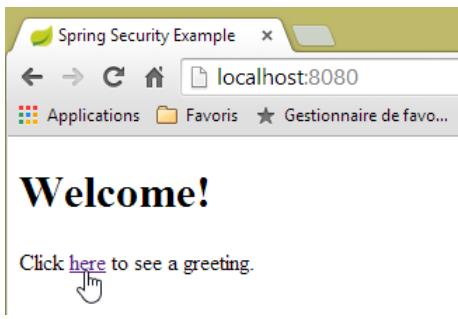
```

1. package hello;
2.
3. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
4. import org.springframework.boot.SpringApplication;
5. import org.springframework.context.annotation.ComponentScan;
6. import org.springframework.context.annotation.Configuration;
7.
8. @EnableAutoConfiguration
9. @Configuration
10. @ComponentScan
11. public class Application {
12.
13.     public static void main(String[] args) throws Throwable {
14.         SpringApplication.run(Application.class, args);
15.     }
16.
17. }
```

- ligne 8 : l'annotation [@EnableAutoConfiguration] demande à Spring Boot (ligne 3) de faire la configuration que le développeur n'aura pas fait explicitement ;
- ligne 9 : fait de la classe [Application] une classe de configuration Spring ;
- ligne 10 : demande le scan du dossier de la classe [Application] afin de rechercher des composants Spring. Les deux classes [MvcConfig] et [WebSecurityConfig] vont être ainsi découvertes car elles ont l'annotation [@Configuration] ;
- ligne 13 : la méthode [main] de la classe exécutable ;
- ligne 14 : la méthode statique [SpringApplication.run] est exécutée avec comme paramètre la classe de configuration [Application]. Nous avons déjà rencontré ce processus et nous savons que le serveur Tomcat embarqué dans les dépendances Maven du projet va être lancé et le projet déployé dessus. Nous avons vu que quatre URL étaient gérées [/ , /home, /login, /hello] et que certaines étaient protégées par des droits d'accès.

2.13.6 Tests de l'application

Commençons par demander l'URL [/] qui est l'une des quatre URL acceptées. Elle est associée à la vue [/templates/home.html] :



L'URL demandée [/] est accessible à tous. C'est pourquoi nous l'avons obtenue. Le lien [here] est le suivant :

```
Click <a href="/hello">here</a> to see a greeting.
```

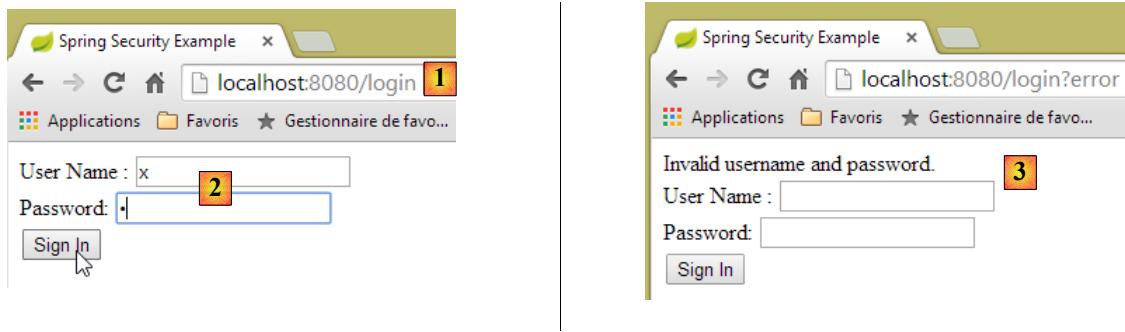
L'URL [/hello] va être demandée lorsqu'on va cliquer sur le lien. Celle-ci est protégée :

URL	règle	code
/, /home	accès sans être authentifié	<code>http.authorizeRequests().antMatchers("/", "/home").permitAll()</code>
autres URL	accès authentifié uniquement	<code>http.anyRequest().authenticated();</code>

Il faut être authentifié pour l'obtenir. Spring Security va alors rediriger le navigateur client vers la page d'authentification. D'après la configuration vue, c'est la page d'URL [/login]. Celle-ci est accessible à tous :

```
http.formLogin().loginPage("/login").permitAll().and().logout().permitAll();
```

Nous l'obtenons donc [1] :



Le code source de la page obtenue est le suivant :

```

1.  <!DOCTYPE html>
2.
3.  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-
   springsecurity3">
4.  ...
5.      <form method="post" action="/login">
6.      ...
7.          <input type="hidden" name="_csrf" value="87bea06a-a177-459d-b279-c6068a7ad3eb" />
8.      </form>
9.  </body>
10. </html>
```

- ligne 7, un champ caché apparaît qui n'est pas dans la page [login.html] d'origine. C'est Thymeleaf qui l'a ajouté. Ce code appelé CSRF (Cross Site Request Forgery) vise à éliminer une faille de sécurité. Ce jeton doit être renvoyé à Spring Security avec l'authentification pour que cette dernière soit acceptée ;

Nous nous souvenons que seul l'utilisateur user/password est reconnu par Spring Security. Si nous entrons autre chose en [2], nous obtenons la même page avec un message d'erreur en [3]. Spring Security a redirigé le navigateur vers l'URL [http://localhost:8080/login?error]. La présence du paramètre [error] a déclenché l'affichage de la balise :

```
<div th:if="${param.error}">Invalid username and password.</div>
```

Maintenant, entrons les valeurs attendues user/password [4] :



- en [4], nous nous identifions ;
- en [5], Spring Security nous redirige vers l'URL [/hello] car c'est l'URL que nous demandions lorsque nous avons été redirigé vers la page de login. L'identité de l'utilisateur a été affichée par la ligne suivante de [hello.html] :

```
<h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
```

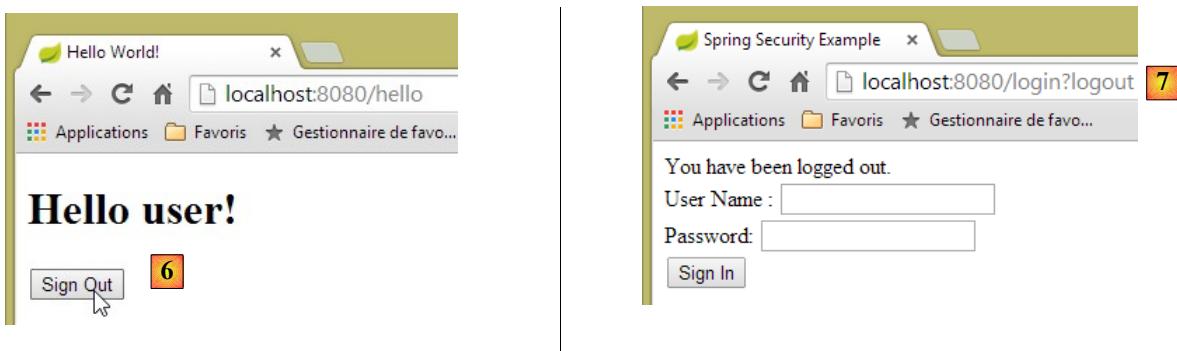
La page [5] affiche le formulaire suivant :

```
1. <form th:action="@{/Logout}" method="post">
2.   <input type="submit" value="Sign Out" />
3. </form>
```

Lorsqu'on clique sur le bouton [Sign Out], un POST va être fait sur l'URL [/logout]. Celle-ci comme l'URL [/login] est accessible à tous :

```
http.formLogin().LoginPage("/login").permitAll().and().logout().permitAll();
```

Dans notre association URL / vues, nous n'avons rien défini pour l'URL [/logout]. Que va-t-il se passer ? Essayons :



- en [6], nous cliquons sur le bouton [Sign Out] ;
- en [7], nous voyons que nous avons été redirigés vers l'URL [http://localhost:8080/login?logout]. C'est Spring Security qui a demandé cette redirection. La présence du paramètre [logout] dans l'URL a fait afficher la ligne suivante de la vue :

```
<div th:if="${param.Logout}">You have been logged out.</div>
```

2.13.7 Conclusion

Dans l'exemple précédent, nous aurions pu écrire l'application web d'abord puis la sécuriser ensuite. Spring Security n'est pas intrusif. On peut mettre en place la sécurité d'une application web déjà écrite. Par ailleurs, nous avons découvert les points suivants :

- il est possible de définir une page d'authentification ;
- l'authentification doit être accompagnée du jeton CSRF délivré par Spring Security ;

- si l'authentification échoue, on est redirigé vers la page d'authentification avec de plus un paramètre **error** dans l'URL ;
- si l'authentification réussit, on est redirigé vers la page demandée lorsque l'autentification a eu lieu. Si on demande directement la page d'authentification sans passer par une page intermédiaire, alors Spring Security nous redirige vers l'URL [/] (ce cas n'a pas été présenté) ;
- on se déconnecte en demandant l'URL [/logout] avec un POST. Spring Security nous redirige alors vers la page d'authentification avec le paramètre **logout** dans l'URL ;

Toutes ces conclusions reposent sur des comportements par défaut de Spring Security. Ces comportements peuvent être changés par configuration en redéfinissant certaines méthodes de la classe **[WebSecurityConfigurerAdapter]**.

Le tutoriel précédent nous aidera peu dans la suite. Nous allons en effet utiliser :

- une base de données pour stocker les utilisateurs, leurs mots de passe et leurs rôles ;
- une authentification par entête HTTP ;

On trouve assez peu de tutoriels pour ce qu'on veut faire ici. La solution qui va être proposée est un assemblage de codes trouvés ici et là.

2.14 Mise en place de la sécurité sur le service web des rendez-vous

2.14.1 La base de données

La base de données [rdvmedecins] évolue pour prendre en compte les utilisateurs, leurs mots de passe et leur rôles. Trois nouvelles tables apparaissent :

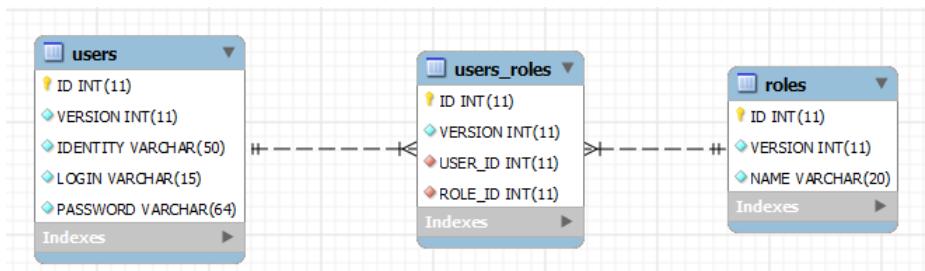


Table [USERS] : les utilisateurs

- ID : clé primaire ;
- VERSION : colonne de versioning de la ligne ;
- IDENTITY : une identité descriptive de l'utilisateur ;
- LOGIN : le login de l'utilisateur ;
- PASSWORD : son mot de passe ;

Dans la table USERS, les mots de passe ne sont pas stockés en clair :

ID	VERSION	IDENTITY	LOGIN	PASSWORD
7	0	guest	guest	\$2a\$10\$Gzyp54mvkgMH0SPQkXo.Zeu.DwJ/QI50PRXLf2FkolMTs7fr6A2J2
8	0	admin	admin	\$2a\$10\$m79V6MKt9GPDdpjSulyqReqUioqYwXy8ollt/.ia15FhX2fym3AE6
9	0	user	user	\$2a\$10\$ph5y/1H89YC11oGVLB49fON.dZwnu44bAOKMK1FFI//xjAvsr/Ese

L'algorithme qui crypte les mots de passe est l'algorithme BCRYPT.

Table [ROLES] : les rôles

- ID : clé primaire ;
- VERSION : colonne de versioning de la ligne ;
- NAME : nom du rôle. Par défaut, Spring Security attend des noms de la forme ROLE_XX, par exemple ROLE_ADMIN ou ROLE_GUEST ;

ID	VERSION	NAME
1	1	ROLE_ADMIN
2	1	ROLE_USER

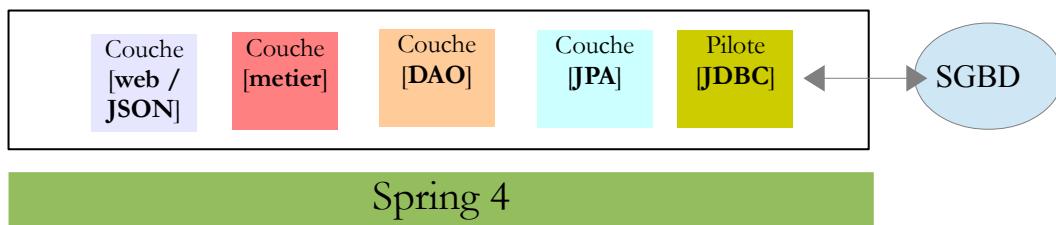
Table [USERS_ROLES] : table de jointure USERS / ROLES

Un utilisateur peut avoir plusieurs rôles, un rôle peut rassembler plusieurs utilisateurs. On a une relation **plusieurs à plusieurs** matérialisée par la table [USERS_ROLES].

- ID : clé primaire ;
- VERSION : colonne de versioning de la ligne ;
- USER_ID : identifiant d'un utilisateur ;
- ROLE_ID : identifiant d'un rôle ;

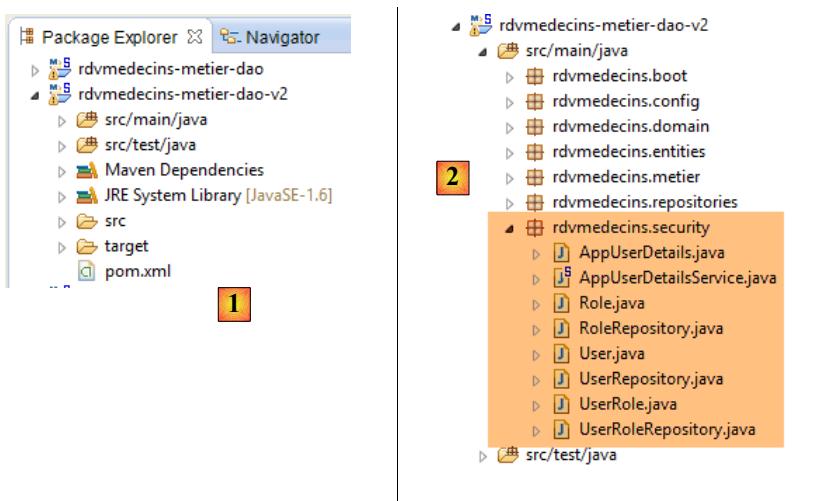
ID	VERSION	USER_ID	ROLE_ID
1	1	1	2
2	1	2	1

Parce que nous modifions la base de données, l'ensemble des couches du projet [métier, DAO, JPA] doit être modifié :



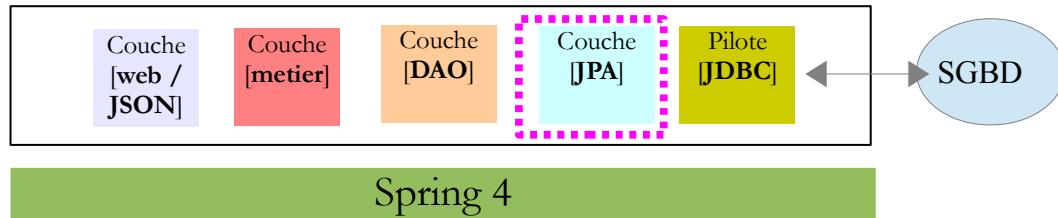
2.14.2 Le nouveau projet Eclipse du [métier, DAO, JPA]

Nous dupliquons le projet initial [rdvmedecins-metier-dao] en [rdvmedecins-metier-dao-v2] :



- en [1] : le nouveau projet ;
- en [2] : les modifications amenées par la prise en compte de la sécurité ont été rassemblées dans un unique paquetage [rdvmedecins.security]. Ces nouveaux éléments appartiennent aux couches [JPA] et [DAO] mais par simplicité je les ai rassemblés dans un même paquetage.

2.14.3 Les nouvelles entités [JPA]



La couche JPA définit trois nouvelles entités :

```
└─ rdvmedecins.security
   └─ AppUserDetails.java
   └─ AppUserDetailsService.java
   └─ Role.java
   └─ RoleRepository.java
   └─ User.java
   └─ UserRepository.java
   └─ UserRole.java
   └─ UserRoleRepository.java
```

La classe [User] est l'image de la table [USERS] :

```
1. package rdvmedecins.entities;
2.
3. import javax.persistence.Column;
4. import javax.persistence.Entity;
5. import javax.persistence.Table;
6.
7. @Entity
8. @Table(name = "USERS")
9. public class User extends AbstractEntity {
10.     private static final long serialVersionUID = 1L;
11.
12.     // propriétés
13.     private String identity;
14.     private String login;
15.     private String password;
16.
17.     // constructeur
18.     public User() {
19.     }
20.
21.     public User(String identity, String login, String password) {
22.         this.identity = identity;
23.         this.login = login;
24.         this.password = password;
25.     }
26.
27.     // identité
28.     @Override
29.     public String toString() {
30.         return String.format("User[%s,%s,%s]", identity, login, password);
31.     }
32.
33.     // getters et setters
34.     ....
35. }
```

- ligne 9 : la classe étend la classe [AbstractEntity] déjà utilisée pour les autres entités ;
- lignes 13-15 : on ne précise pas de nom pour les colonnes parce qu'elles portent le même nom que les champs qui leur sont associés ;

La classe [Role] est l'image de la table [ROLES] :

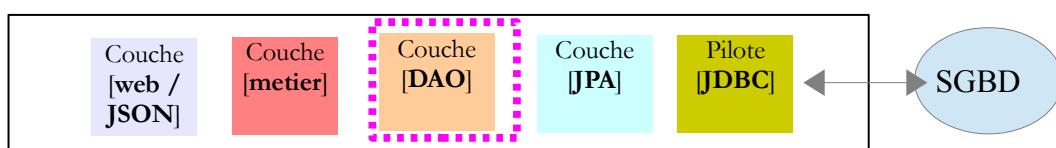
```
1. package rdvmedecins.entities;
2.
3. import javax.persistence.Column;
4. import javax.persistence.Entity;
5. import javax.persistence.Table;
6.
7. @Entity
8. @Table(name = "ROLES")
9. public class Role extends AbstractEntity {
10.
11.     private static final long serialVersionUID = 1L;
12.
13.     // propriétés
14.     private String name;
15.
16.     // constructeurs
17.     public Role() {
18.     }
19.
20.     public Role(String name) {
21.         this.name = name;
22.     }
23.
24.     // identité
25.     @Override
26.     public String toString() {
27.         return String.format("Role[%s]", name);
28.     }
29.
30.     // getters et setters
31. ...
32. }
```

La classe [UserRole] est l'image de la table [USERS_ROLES] :

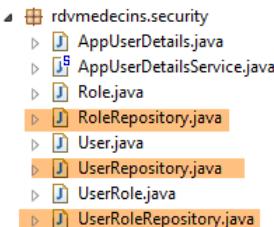
```
1. package rdvmedecins.entities;
2.
3. import javax.persistence.Entity;
4. import javax.persistence.JoinColumn;
5. import javax.persistence.ManyToOne;
6. import javax.persistence.Table;
7.
8. @Entity
9. @Table(name = "USERS_ROLES")
10. public class UserRole extends AbstractEntity {
11.
12.     private static final long serialVersionUID = 1L;
13.
14.     // un UserRole référence un User
15.     @ManyToOne
16.     @JoinColumn(name = "USER_ID")
17.     private User user;
18.     // un UserRole référence un Role
19.     @ManyToOne
20.     @JoinColumn(name = "ROLE_ID")
21.     private Role role;
22.
23.     // getters et setters
24. ...
25. }
```

- lignes 15-17 : matérialisent la clé étrangère de la table [USERS_ROLES] vers la table [USERS] ;
- lignes 19-21 : matérialisent la clé étrangère de la table [USERS_ROLES] vers la table [ROLES] ;

2.14.4 Modifications de la couche [DAO]



La couche [DAO] s'enrichit de trois nouveaux [Repository] :



L'interface [UserRepository] gère les accès aux entités [User] :

```
1. package rdvmedecins.repositories;
2.
3. import org.springframework.data.jpa.repository.Query;
4. import org.springframework.data.repository.CrudRepository;
5.
6. import rdvmedecins.entities.Role;
7. import rdvmedecins.entities.User;
8.
9. public interface UserRepository extends CrudRepository<User, Long> {
10.
11.     // liste des rôles d'un utilisateur identifié par son id
12.     @Query("select ur.role from UserRole ur where ur.user.id=?1")
13.     Iterable<Role> getRoles(long id);
14.
15.     // liste des rôles d'un utilisateur identifié par son login et son mot de passe
16.     @Query("select ur.role from UserRole ur where ur.user.login=?1 and ur.user.password=?2")
17.     Iterable<Role> getRoles(String login, String password);
18.
19.     // recherche d'un utilisateur via son login
20.     User findUserByLogin(String login);
21. }
```

- ligne 9 : l'interface [UserRepository] étend la l'interface [CrudRepository] de Spring Data (ligne 4) ;
- lignes 12-13 : la méthode [getRoles(User user)] permet d'avoir tous les rôles d'un utilisateur identifié par son [id]
- lignes 16-17 : idem mais pour un utilisateur identifié pas ses login / mot de passe ;

L'interface [RoleRepository] gère les accès aux entités [Role] :

```
1. package rdvmedecins.security;
2.
3. import org.springframework.data.repository.CrudRepository;
4.
5. public interface RoleRepository extends CrudRepository<Role, Long> {
6.
7.     // recherche d'un rôle via son nom
8.     Role findRoleByName(String name);
9.
10. }
```

- ligne 5 : l'interface [RoleRepository] étend l'interface [CrudRepository] ;
- ligne 8 : on peut chercher un rôle via son nom ;

L'interface [userRoleRepository] gère les accès aux entités [UserRole] :

```
1. package rdvmedecins.security;
2.
3. import org.springframework.data.repository.CrudRepository;
4.
5. public interface UserRoleRepository extends CrudRepository<UserRole, Long> {
6.
7. }
```

- ligne 5 : l'interface [UserRoleRepository] se contente d'étendre l'interface [CrudRepository] sans lui ajouter de nouvelles méthodes ;

2.14.5 Les classes de gestion des utilisateurs et des rôles

```

    ▲ rdvmedecins.security
    ▷ AppUserDetails.java
    ▷ AppUserService.java
    ▷ Role.java
    ▷ RoleRepository.java
    ▷ User.java
    ▷ UserRepository.java
    ▷ UserRole.java
    ▷ UserRoleRepository.java
  
```

Spring Security impose la création d'une classe implémentant l'interface [UsersDetail] suivante :

Method Summary

<code>Collection<GrantedAuthority></code>	<code>getAuthorities()</code> Returns the authorities granted to the user.
<code>String</code>	<code> getPassword()</code> Returns the password used to authenticate the user.
<code>String</code>	<code> getUsername()</code> Returns the username used to authenticate the user.
<code>boolean</code>	<code>isAccountNonExpired()</code> Indicates whether the user's account has expired.
<code>boolean</code>	<code>isAccountNonLocked()</code> Indicates whether the user is locked or unlocked.
<code>boolean</code>	<code>isCredentialsNonExpired()</code> Indicates whether the user's credentials (password) has expired.
<code>boolean</code>	<code>isEnabled()</code> Indicates whether the user is enabled or disabled.

Cette interface est ici implémentée par la classe [AppUserDetails] :

```

1. package rdvmedecins.security;
2.
3. import java.util.ArrayList;
4. import java.util.Collection;
5.
6. import org.springframework.security.core.GrantedAuthority;
7. import org.springframework.security.core.authority.SimpleGrantedAuthority;
8. import org.springframework.security.core.userdetails.UserDetails;
9.
10. public class AppUserDetails implements UserDetails {
11.
12.     private static final long serialVersionUID = 1L;
13.
14.     // propriétés
15.     private User user;
16.     private UserRepository userRepository;
17.
18.     // constructeurs
19.     public AppUserDetails() {
20.     }
21.
22.     public AppUserDetails(User user, UserRepository userRepository) {
23.         this.user = user;
  
```

```

24.     this.userRepository = userRepository;
25. }
26.
27. // -----interface
28. @Override
29. public Collection<? extends GrantedAuthority> getAuthorities() {
30.     Collection<GrantedAuthority> authorities = new ArrayList<>();
31.     for (Role role : userRepository.getRoles(user.getId())) {
32.         authorities.add(new SimpleGrantedAuthority(role.getName()));
33.     }
34.     return authorities;
35. }
36.
37. @Override
38. public String getPassword() {
39.     return user.getPassword();
40. }
41.
42. @Override
43. public String getUsername() {
44.     return user.getLogin();
45. }
46.
47. @Override
48. public boolean isAccountNonExpired() {
49.     return true;
50. }
51.
52. @Override
53. public boolean isAccountNonLocked() {
54.     return true;
55. }
56.
57. @Override
58. public boolean isCredentialsNonExpired() {
59.     return true;
60. }
61.
62. @Override
63. public boolean isEnabled() {
64.     return true;
65. }
66.
67. // getters et setters
68. ...
69. }

```

- ligne 10 : la classe [AppUserDetails] implémente l'interface [UserDetails] ;
- lignes 15-16 : la classe encapsule un utilisateur (ligne 15) et le repository qui permet d'avoir les détails de cet utilisateur (ligne 16) ;
- lignes 22-25 : le constructeur qui instancie la classe avec un utilisateur et son repository ;
- lignes 28-35 : implémentation de la méthode [getAuthorities] de l'interface [UserDetails]. Elle doit construire une collection d'éléments de type [GrantedAuthority] ou dérivé. Ici, nous utilisons le type dérivé [SimpleGrantedAuthority] (ligne 32) qui encapsule le nom d'un des rôles de l'utilisateur de la ligne 15 ;
- lignes 31-33 : on parcourt la liste des rôles de l'utilisateur de la ligne 15 pour construire une liste d'éléments de type [SimpleGrantedAuthority] ;
- lignes 38-40 : implémentent la méthode [getPassword] de l'interface [UserDetails]. On rend le mot de passe de l'utilisateur de la ligne 15 ;
- lignes 38-40 : implémentent la méthode [getUserName] de l'interface [UserDetails]. On rend le login de l'utilisateur de la ligne 15 ;
- lignes 47-50 : le compte de l'utilisateur n'expire jamais ;
- lignes 52-55 : le compte de l'utilisateur n'est jamais bloqué ;
- lignes 57-60 : les identifiants de l'utilisateur n'expirent jamais ;
- lignes 62-65 : le compte de l'utilisateur est toujours actif ;

Spring Security impose également l'existence d'une classe implémentant l'interface [**AppUserDetailsService**] :

Method Summary

UserDetails

`loadUserByUsername(String username)`

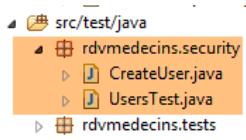
Locates the user based on the username.

Cette interface est implémentée par la classe [AppUserDetails] suivante :

```
1. package rdvmedecins.security;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.security.core.userdetails.UserDetails;
5. import org.springframework.security.core.userdetails.UserDetailsService;
6. import org.springframework.security.core.userdetails.UsernameNotFoundException;
7. import org.springframework.stereotype.Service;
8.
9. @Service
10. public class AppUserDetailsService implements UserDetailsService {
11.
12.     @Autowired
13.     private UserRepository userRepository;
14.
15.     @Override
16.     public UserDetails loadUserByUsername(String login) throws UsernameNotFoundException {
17.         // on cherche l'utilisateur via son login
18.         User user = userRepository.findUserByLogin(login);
19.         // trouvé ?
20.         if (user == null) {
21.             throw new UsernameNotFoundException(String.format("login [%s] inexistant", login));
22.         }
23.         // on rend les détails de l'utilisateur
24.         return new AppUserDetails(user, userRepository);
25.     }
26.
27. }
```

- ligne 9 : la classe sera un composant Spring, donc disponible dans son contexte ;
- lignes 12-13 : le composant [UserRepository] sera injecté ici ;
- lignes 16-25 : implémentation de la méthode [loadUserByUsername] de l'interface [UserDetailsService] (ligne 10). Le paramètre est le login de l'utilisateur ;
- ligne 18 : l'utilisateur est recherché via son login ;
- lignes 20-22 : s'il n'est pas trouvé, une exception est lancée ;
- ligne 24 : un objet [AppUserDetails] est construit et rendu. Il est bien de type [UserDetails] (ligne 16) ;

2.14.6 Tests de la couche [DAO]



Tout d'abord, nous créons une classe exécutable [CreateUser] capable de créer un utilisateur avec un rôle :

```
1. package rdvmedecins.security;
2.
3. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4. import org.springframework.security.crypto.BCrypt;
5.
6. import rdvmedecins.config.DomainAndPersistenceConfig;
7. import rdvmedecins.security.Role;
8. import rdvmedecins.security.RoleRepository;
9. import rdvmedecins.security.User;
```

```

10. import rdvmedecins.security.UserRepository;
11. import rdvmedecins.security.UserRole;
12. import rdvmedecins.security.UserRoleRepository;
13.
14. public class CreateUser {
15.
16.     public static void main(String[] args) {
17.         // syntaxe : login password roleName
18.
19.         // il faut trois paramètres
20.         if (args.length != 3) {
21.             System.out.println("Syntaxe : [pg] user password role");
22.             System.exit(0);
23.         }
24.         // on récupère les paramètres
25.         String login = args[0];
26.         String password = args[1];
27.         String roleName = String.format("ROLE_%s", args[2].toUpperCase());
28.         // contexte Spring
29.         AnnotationConfigApplicationContext context = new
30.             AnnotationConfigApplicationContext(DomainAndPersistenceConfig.class);
31.         UserRepository userRepository = context.getBean(UserRepository.class);
32.         RoleRepository roleRepository = context.getBean(RoleRepository.class);
33.         UserRoleRepository userRoleRepository = context.getBean(UserRoleRepository.class);
34.         // le rôle existe-t-il déjà ?
35.         Role role = roleRepository.findRoleByName(roleName);
36.         // s'il n'existe pas on le crée
37.         if (role == null) {
38.             role = roleRepository.save(new Role(roleName));
39.         }
40.         // l'utilisateur existe-t-il déjà ?
41.         User user = userRepository.findUserByLogin(login);
42.         // s'il n'existe pas on le crée
43.         if (user == null) {
44.             // on hashe le mot de passe avec bcrypt
45.             String crypt = BCrypt.hashpw(password, BCrypt.gensalt());
46.             // on sauvegarde l'utilisateur
47.             user = userRepository.save(new User(login, login, crypt));
48.             // on crée la relation avec le rôle
49.             userRoleRepository.save(new UserRole(user, role));
50.         } else {
51.             // l'utilisateur existe déjà- a-t-il le rôle demandé ?
52.             boolean trouvé = false;
53.             for (Role r : userRepository.getRoles(user.getId())) {
54.                 if (r.getName().equals(roleName)) {
55.                     trouvé = true;
56.                     break;
57.                 }
58.             }
59.             // si pas trouvé, on crée la relation avec le rôle
60.             if (!trouvé) {
61.                 userRoleRepository.save(new UserRole(user, role));
62.             }
63.         }
64.         // fermeture contexte Spring
65.         context.close();
66.     }
67.
68. }
```

- ligne 17 : la classe attend trois arguments définissant un utilisateur : son login, son mot de passe, son rôle ;
- lignes 25-27 : les trois paramètres sont récupérés ;
- ligne 29 : le contexte Spring est construit à partir de la classe de configuration [DomainAndPersistenceConfig]. Cette classe existait déjà dans le projet précédent. Elle doit évoluer de la façon suivante :

```

1. @EnableJpaRepositories(basePackages = { "rdvmedecins.repositories", "rdvmedecins.security" })
2. @EnableAutoConfiguration
3. @ComponentScan(basePackages = { "rdvmedecins" })
4. @EntityScan(basePackages = { "rdvmedecins.entities", "rdvmedecins.security" })
5. @EnableTransactionManagement
6. public class DomainAndPersistenceConfig {
7. ....
```

8. }

- ligne 1 : il faut indiquer qu'il y a maintenant des composants [Repository] dans le paquetage [rdvmedecins.security] ;
- ligne 4 : il faut indiquer qu'il y a maintenant des entités JPA dans le paquetage [rdvmedecins.security] ;

Revenons au code de création d'un utilisateur :

- lignes 30-32 : on récupère les références des trois [Repository] qui peuvent nous être utiles pour créer l'utilisateur ;
- ligne 34 : on regarde si le rôle existe déjà ;
- lignes 36-38 : si ce n'est pas le cas, on le crée en base. Il aura un nom du type [ROLE_XX] ;
- ligne 40 : on regarde si le login existe déjà ;
- lignes 42-49 : si le login n'existe pas, on le crée en base ;
- ligne 44 : on crypte le mot de passe. On utilise ici, la classe [BCrypt] de Spring Security (ligne 4). On a donc besoin des archives de ce framework. Le fichier [pom.xml] inclut une nouvelle dépendance :

```
1. <dependency>
2.   <groupId>org.springframework.boot</groupId>
3.   <artifactId>spring-boot-starter-security</artifactId>
4. </dependency>
```

- ligne 46 : l'utilisateur est persisté en base ;
- ligne 48 : ainsi que la relation qui le lie à son rôle ;
- lignes 51-57 : cas où le login existe déjà – on regarde alors si parmi ses rôles se trouve déjà le rôle qu'on veut lui attribuer ;
- ligne 59-61 : si le rôle cherché n'a pas été trouvé, on crée une ligne dans la table [USERS_ROLES] pour relier l'utilisateur à son rôle ;
- on ne s'est pas protégé des exceptions éventuelles. C'est une classe de soutien pour créer rapidement un utilisateur avec un rôle.

Lorsqu'on exécute la classe avec les arguments [x x guest], on obtient en base les résultats suivants :

Table [USERS]

ID	VERSION	IDENTITY	LOGIN	PASSWORD
7	0	guest	guest	\$2a\$10\$Gzyp54mvkgMH0SPQkXo.Zeu.DvJ/Ql50PRXLf2FkolMTs7fr6A2J2
8	0	admin	admin	\$2a\$10\$m79V6MkI9GPDDpjSulyqReqUioqYwXy8ollt/.ia15FhX2fym3AE6
9	0	user	user	\$2a\$10\$ph5y/1H89YC11oGVLB49fON.dZwnu44bAOKMK1FFI//xjAvsr/Ese
12	0	x	x	\$2a\$10\$dAKd2SuQpIR1iFhoBUUFs.XiA0lYxNqOmrv97Gbr5KBoHzEi/5HG

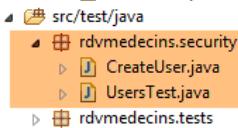
Table [ROLES]

ID	VERSION	NAME
5	0	ROLE_GUEST
6	0	ROLE_ADMIN
7	0	ROLE_USER

Table [USERS_ROLES]

ID	VERSION	USER_ID	ROLE_ID
4	0	7	5
5	0	8	6
6	0	9	7
9	0	12	5

Considérons maintenant la seconde classe [UsersTest] qui est un test JUnit :



```
1. package rdvmedecins.security;
2.
3. import java.util.List;
4.
5. import org.junit.Assert;
6. import org.junit.Test;
7. import org.junit.runner.RunWith;
8. import org.springframework.beans.factory.annotation.Autowired;
9. import org.springframework.boot.test.SpringApplicationConfiguration;
10. import org.springframework.core.authority.SimpleGrantedAuthority;
11. import org.springframework.security.crypto.bcrypt.BCrypt;
12. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
13.
14. import rdvmedecins.config.DomainAndPersistenceConfig;
15.
16. import com.google.common.collect.Lists;
17.
18. @SpringApplicationConfiguration(classes = DomainAndPersistenceConfig.class)
19. @RunWith(SpringJUnit4ClassRunner.class)
20. public class UsersTest {
21.
22.     @Autowired
23.     private UserRepository userRepository;
24.     @Autowired
25.     private AppUserDetailsService appUserDetailsService;
26.
27.     @Test
28.     public void findAllUsersWithTheirRoles() {
29.         Iterable<User> users = userRepository.findAll();
30.         for (User user : users) {
31.             System.out.println(user);
32.             display("Roles :", userRepository.getRoles(user.getId()));
33.         }
34.     }
35.
36.     @Test
37.     public void findUserByLogin() {
38.         // on récupère l'utilisateur [admin]
39.         User user = userRepository.findUserByLogin("admin");
40.         // on vérifie que son mot de passe est [admin]
41.         Assert.assertTrue(BCrypt.checkpw("admin", user.getPassword()));
42.         // on vérifie le rôle de admin / admin
43.         List<Role> roles = Lists.newArrayList(userRepository.getRoles("admin", user.getPassword()));
44.         Assert.assertEquals(1L, roles.size());
45.         Assert.assertEquals("ROLE_ADMIN", roles.get(0).getName());
46.     }
47.
48.     @Test
49.     public void loadUserByUsername() {
50.         // on récupère l'utilisateur [admin]
51.         AppUserDetails userDetails = (AppUserDetails) appUserDetailsService.loadUserByUsername("admin");
52.         // on vérifie que son mot de passe est [admin]
53.         Assert.assertTrue(BCrypt.checkpw("admin", userDetails.getPassword()));
54.         // on vérifie le rôle de admin / admin
55.         @SuppressWarnings("unchecked")
56.         List<SimpleGrantedAuthority> authorities = (List<SimpleGrantedAuthority>)
57.             userDetails.getAuthorities();
58.         Assert.assertEquals(1L, authorities.size());
59.         Assert.assertEquals("ROLE_ADMIN", authorities.get(0).getAuthority());
60.     }
61.     // méthode utilitaire - affiche les éléments d'une collection
62.     private void display(String message, Iterable<?> elements) {
63.         System.out.println(message);
```

```

64.     for (Object element : elements) {
65.         System.out.println(element);
66.     }
67. }
68. }
```

- lignes 27-34 : test visuel. On affiche tous les utilisateurs avec leurs rôles ;
- lignes 36-46 : on vérifie que l'utilisateur [admin] a le mot de passe [admin] et le rôle [ROLE_ADMIN] en utilisant le repository [UserRepository] ;
- ligne 41 : [admin] est le mot de passe en clair. En base, il est crypté selon l'algorithme BCrypt. La méthode [BCrypt.checkpw] permet de vérifier que le mot de passe en clair une fois crypté est bien égal à celui qui est en base ;
- lignes 48-59 : on vérifie que l'utilisateur [admin] a le mot de passe [admin] et le rôle [ROLE_ADMIN] en utilisant le service [appUserDetailsService] ;

L'exécution des tests réussit avec les logs suivants :

```

1. User[guest, guest,$2a$10$Gzyp54mvkgMH0SPQkXo.Zeu.DvJ/Q150PRXLf2Fko1MTs7fr6A2J2]
2. Roles :
3. Role[ROLE_GUEST]
4. User[admin, admin,$2a$10$m79V6MKt9GPDdpjSulyqReqUioqYwXy8o1lt/.ia15FhX2fyM3AE6]
5. Roles :
6. Role[ROLE_ADMIN]
7. User[user, user,$2a$10$ph5y/1H89YC11oGVLB49f0N.dZwnu44bA0KMK1FF1//xjAvsr/Ese]
8. Roles :
9. Role[ROLE_USER]
10. User[x, x,$2a$10$dAKd2SuQplR1iFhoBUUFs.XiA01YxNq0mrkv97Gbr5KBoHzEi/5HG]
11. Roles :
12. Role[ROLE_GUEST]
```

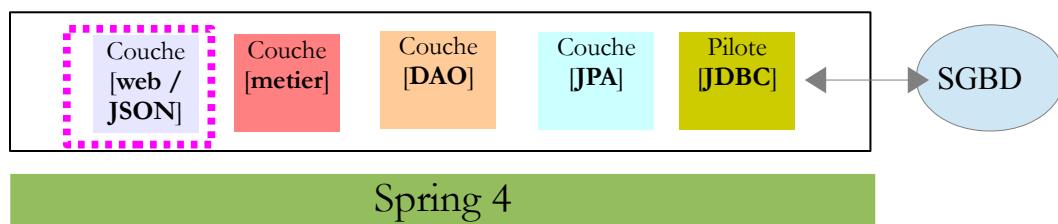
2.14.7 Conclusion intermédiaire

L'ajout des classes nécessaires à Spring Security a pu se faire avec peu de modifications du projet original. Rappelons-les :

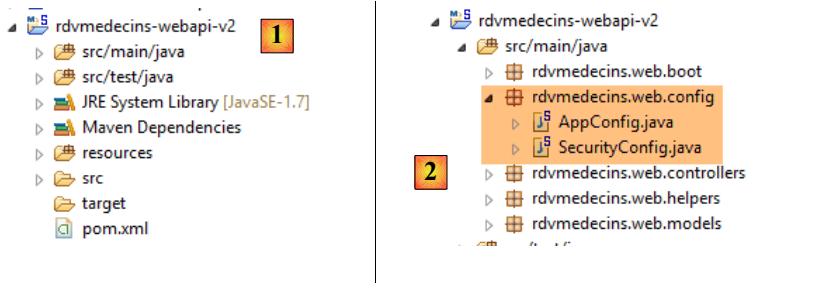
- ajout d'une dépendance sur Spring Security dans le fichier [pom.xml] ;
- création de trois tables supplémentaires dans la base de données ;
- création d'entités JPA et de composants Spring dans le package [rdvmedecins.security] ;

Ce cas très favorable découle du fait que les trois tables ajoutées dans la base de données sont indépendantes des tables existantes. On aurait même pu les mettre dans une base de données séparée. Ceci a été possible parce qu'on a décidé qu'un utilisateur avait une existence indépendante des médecins et des clients. Si ces derniers avaient été des utilisateurs potentiels, il aurait fallu créer des liens entre la table [USERS] et les tables [MEDECINS] et [CLIENTS]. Cela aurait eu alors un impact important sur le projet existant.

2.14.8 Le projet Eclipse de la couche [web]



Le projet [rdvmedecins-webapi] précédent est dupliqué dans le projet [rdvmedecins-webapi-v2] [1] :



Les seules modifications sont à faire dans le package [rdvmedecins.web.config] où il faut configurer Spring Security. Nous avons déjà rencontré une classe de configuration de Spring Security :

```

1. package hello;
2.
3. import org.springframework.context.annotation.Configuration;
4. import
5.     org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
6. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
7. import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
8. import org.springframework.security.config.annotation.web.servlet.configuration.EnableWebMvcSecurity;
9.
10. @Configuration
11. @EnableWebMvcSecurity
12. public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
13.     @Override
14.     protected void configure(HttpSecurity http) throws Exception {
15.         http.authorizeRequests().antMatchers("/", "/home").permitAll().anyRequest().authenticated();
16.         http.formLogin().loginPage("/login").permitAll().and().logout().permitAll();
17.     }
18.     @Override
19.     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
20.         auth.inMemoryAuthentication().withUser("user").password("password").roles("USER");
21.     }
22. }
```

Nous allons suivre la même démarche :

- ligne 11 : définir une classe qui étend la classe [WebSecurityConfigurerAdapter] ;
- ligne 13 : définir une méthode [configure(HttpSecurity http)] qui définit les droits d'accès aux différentes URL du service web ;
- ligne 19 : définir une méthode [configure(AuthenticationManagerBuilder auth)] qui définit les utilisateurs et leurs rôles ;

La configuration de Spring Security est assurée par la classe [**SecurityConfig**] :

```

1. package rdvmedecins.web.config;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
5. import org.springframework.http.HttpMethod;
6. import
7.     org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
8. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
9. import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
10. import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
11.
12. import rdvmedecins.security.AppUserDetailsService;
13.
14. @EnableAutoConfiguration
15. @EnableWebSecurity
16. public class SecurityConfig extends WebSecurityConfigurerAdapter {
17.     @Autowired
18.     private AppUserDetailsService appUserDetailsService;
19.
20.     @Override
21.     protected void configure(AuthenticationManagerBuilder registry) throws Exception {
22.         // l'authentification est faite par le bean [appUserDetailsService]
```

```

23.      // le mot de passe est crypté par l'algorithme de hachage Bcrypt
24.      registry.userDetailsService(appUserDetailsService).passwordEncoder(new BCryptPasswordEncoder());
25.  }
26.
27.  @Override
28.  protected void configure(HttpSecurity http) throws Exception {
29.      // CSRF
30.      http.csrf().disable();
31.      // le mot de passe est transmis par le header Authorization: Basic xxxx
32.      http.httpBasic();
33.      // seul le rôle ADMIN peut utiliser l'application
34.      http.authorizeRequests() //
35.          .antMatchers("/", "/**") // toutes les URL
36.          .hasRole("ADMIN");
37.  }
38. }
```

- lignes 14-15 : on a repris les annotations de l'exemple ;
- lignes 17-18 : la classe [AppUserDetails] qui donne accès aux utilisateurs de l'application est injectée ;
- lignes 20-21 : la méthode [configure(HttpSecurity http)] définit les utilisateurs et leurs rôles. Elle reçoit en paramètre un type [AuthenticationManagerBuilder]. Ce paramètre est enrichi de deux informations :
 - une référence sur le service [appUserDetailsService] de la ligne 18 qui donne accès aux utilisateurs enregistrés. On notera ici que le fait qu'ils soient enregistrés dans une base de données n'apparaît pas. Ils pourraient donc être dans un cache, délivrés par un service web, ...
 - le type de cryptage utilisé pour le mot de passe. On rappelle ici que nous avons utilisé l'algorithme BCrypt ;
- lignes 27-40 : la méthode [configure(HttpSecurity http)] définit les droits d'accès aux URL du service web ;
- ligne 30 : nous avons vu dans le projet d'introduction que par défaut Spring Security gérait un jeton CSRF (Cross Site Request Forgery) que l'utilisateur qui voulait s'authentifier devait renvoyer au serveur. Ici ce mécanisme est désactivé ;
- ligne 32 : on active le mode d'authentification par entête HTTP. Le client devra envoyer l'entête HTTP suivant :

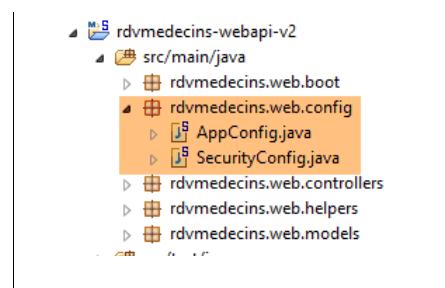
Authorization:Basic code

où code est le codage de la chaîne **login:password** par l'algorithme Base64. Par exemple, le codage Base64 de la chaîne **admin:admin** est **YWRtaW46YWRtaW4=**. Donc l'utilisateur de login [admin] et de mot de passe [admin] enverra l'entête HTTP suivant pour s'authentifier :

Authorization:Basic YWRtaW46YWRtaW4=

- lignes 34-36 : indiquent que toutes les URL du service web sont accessibles aux utilisateurs ayant le rôle [ROLE_ADMIN]. Cela veut dire qu'un utilisateur n'ayant pas ce rôle ne peut accéder au service web ;

La classe [AppConfig] qui configure l'ensemble de l'application évolue comme suit :



```

1. package rdvmedecins.web.config;
2.
3. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Import;
6.
7. import rdvmedecins.config.DomainAndPersistenceConfig;
8.
9. @EnableAutoConfiguration
10. @ComponentScan(basePackages = { "rdvmedecins.web" })
11. @Import({ DomainAndPersistenceConfig.class, SecurityConfig.class })
12. public class AppConfig {
13.
```

14. }

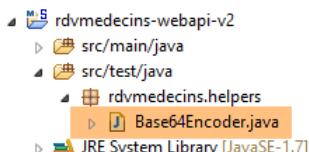
- la modification a lieu ligne 11 : on indique qu'il y a maintenant deux fichiers de configuration à exploiter [DomainAndPersistenceConfig] et [SecurityConfig].

2.14.9 Tests du service web

Nous allons tester le service web avec le client Chrome [Advanced Rest Client]. Nous allons avoir besoin de préciser l'entête HTTP d'authentification :

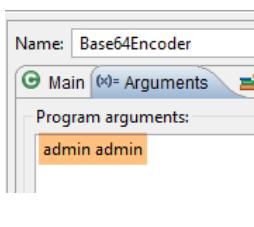
Authorization:Basic code

où [code] est le code Base64 de la chaîne [login:password]. Pour générer ce code, on peut utiliser le programme suivant :



```
1. package rdvmedecins.helpers;
2.
3. import org.springframework.security.crypto.codec.Base64;
4.
5. public class Base64Encoder {
6.
7.     public static void main(String[] args) {
8.         // on attend deux arguments : login password
9.         if (args.length != 2) {
10.             System.out.println("Syntaxe : login password");
11.             System.exit(0);
12.         }
13.         // on récupère les deux arguments
14.         String chaîne = String.format("%s:%s", args[0], args[1]);
15.         // on encode la chaîne
16.         byte[] data = Base64.encode(chaîne.getBytes());
17.         // on affiche son encodage Base64
18.         System.out.println(new String(data));
19.     }
20.
21. }
```

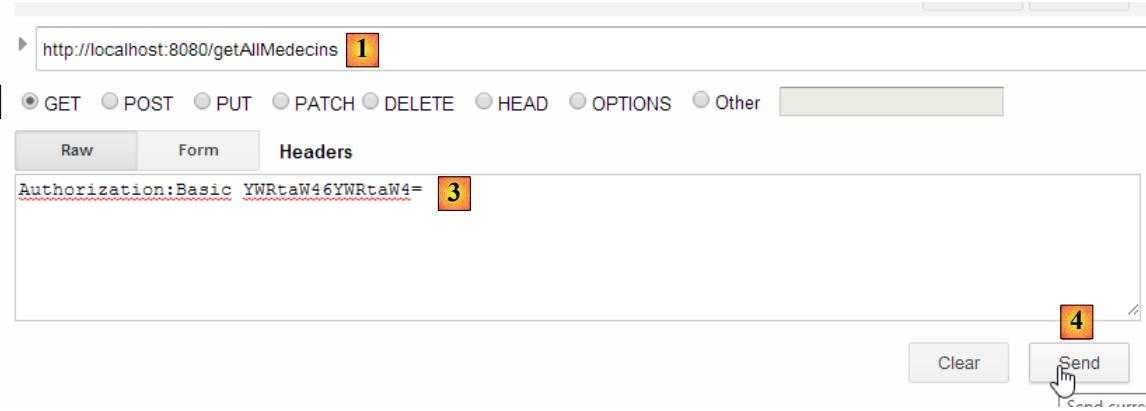
Si nous exécutons ce programme avec les deux arguments [admin admin] :



nous obtenons le résultat suivant :

YWRTawW46YWRTawW4=

Maintenant que nous savons générer l'entête HTTP d'authentification, nous lançons le service web maintenant sécurisé. Puis avec le client Chrome [Advanced Rest Client], nous demandons la liste des tous les médecins :



- en [1], nous demandons l'URL des médecins ;
- en [2], avec une méthode GET ;
- en [3], nous donnons l'entête HTTP de l'authentification. Le code [YWRtaW46YWRtaW4=] est le codage Base64 de la chaîne [admin:admin] ;
- en [4], nous envoyons la commande HTTP ;

La réponse du serveur est la suivante :

Status	200 OK ⓘ Loading time: 1171 ms
Request headers	User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Authorization: Basic YWRtaW46YWRtaW4= 1 Accept: */* Accept-Encoding: gzip,deflate,sdch Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4 Cookie: JSESSIONID=BE74923E59002D698C14BA7A329C6603
Response headers	Server: Apache-Coyote/1.1 ⓘ X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache ⓘ Expires: 0 ⓘ X-Frame-Options: DENY Set-Cookie: JSESSIONID=5EA7ADF025E64A37B0F2019057CA45BC; Path=/; HttpOnly ⓘ Content-Type: application/json;charset=UTF-8 2 Transfer-Encoding: chunked ⓘ Date: Tue, 17 Jun 2014 11:29:28 GMT ⓘ Cache-Control: no-cache, no-store, max-age=0, must-revalidate ⓘ Access-Control-Allow-Origin: chrome-extension://hgmlloofddfdnphfgcellkdfbfbjeloo Access-Control-Allow-Credentials: true Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept, Cookie Access-Control-Allow-Methods: POST, GET, OPTIONS Access-Control-Max-Age: 1998000
Raw	JSON Response
Copy to clipboard Save as file	
<pre>{ status: 0 -data: [4] -0: { id: 1 version: 1 titre: "Mme" nom: "PELISSIER" prenom: "Marie" } -1: { id: 2 version: 1 } }</pre> 3	

- en [1], l'entête HTTP d'authentification ;

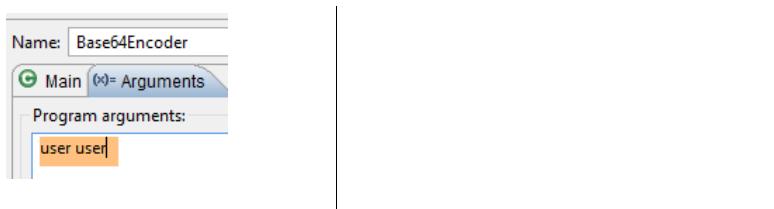
- en [2], le serveur renvoie une réponse JSON ;
- en [3], la liste des médecins.

Tentons maintenant une requête HTTP avec un entête d'authentification incorrect. La réponse est alors la suivante :

The screenshot shows a network request in a browser's developer tools. The URL is `http://localhost:8080/getAllMedecins`. The method is set to GET. The Headers section shows an `Authorization: Basic X` header, which is highlighted with a yellow box and labeled [1]. The status of the response is `401 Unauthorized`, indicated by a yellow box and labeled [2]. The response headers include `User-Agent`, `Accept`, `Accept-Encoding`, `Accept-Language`, and `Cookie`. The response body contains a large amount of server configuration information, including `Server`, `X-Content-Type-Options`, `X-XSS-Protection`, `Cache-Control`, `Pragma`, `Expires`, `X-Frame-Options`, `WWW-Authenticate`, `Content-Type`, `Content-Language`, `Content-Length`, `Date`, `Cache-Control`, `Access-Control-Allow-Origin`, `Access-Control-Allow-Credentials`, `Access-Control-Allow-Headers`, `Access-Control-Allow-Methods`, and `Access-Control-Max-Age`.

- en [1] et [3] : l'entête HTTP d'authentification ;
- en [2] : la réponse du service web ;

Maintenant, essayons l'utilisateur user / user. Il existe mais n'a pas accès au service web. Si nous exécutons le programme d'encodage Base64 avec les deux arguments [user user] :



nous obtenons le résultat suivant :

```
dXNlcjplc2Vy
```

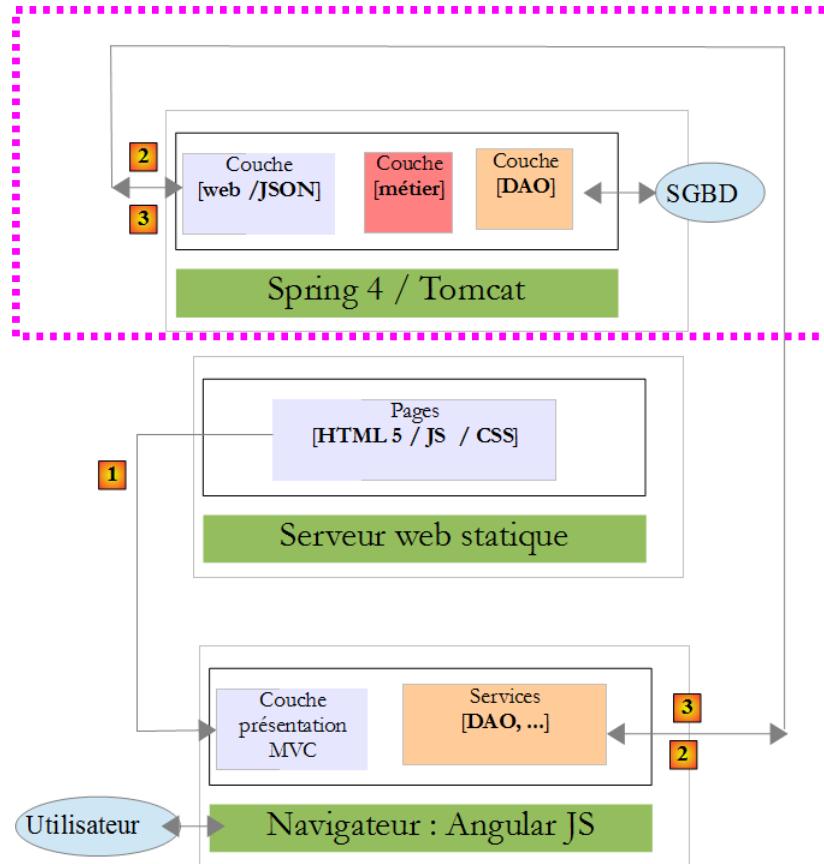
The screenshot shows a browser developer tools Network tab with the following details:

- Request Headers:**
 - Authorization: Basic dXNlcjp1c2Vy [1]
- Status:** 403 Forbidden [2]
- Request Headers (continued):**
 - User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36 [3]
 - Accept: */*
 - Accept-Encoding: gzip,deflate,sdch
 - Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
 - Cookie: JSESSIONID=5EA7ADF025E64A37B0F2019057CA-
- Response Headers:**
 - Server: Apache-Coyote/1.1
 - X-Content-Type-Options: nosniff
 - X-XSS-Protection: 1; mode=block
 - Cache-Control: no-cache, no-store, max-age=0, must-revalidate
 - Pragma: no-cache
 - Expires: 0
 - X-Frame-Options: DENY
 - Set-Cookie: JSESSIONID=090A058686426009961C0062F6
 - Content-Type: text/html;charset=utf-8
 - Content-Language: en
 - Content-Length: 989
 - Date: Tue, 17 Jun 2014 15:45:58 GMT
 - Cache-Control: no-cache, no-store, max-age=0, must-revalidate
 - Access-Control-Allow-Origin: chrome-extension://hgmlloofd
 - Access-Control-Allow-Credentials: true
 - Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type
 - Access-Control-Allow-Methods: POST, GET, OPTIONS
 - Access-Control-Max-Age: 1998000

- en [1] et [3] : l'en-tête HTTP d'authentification ;
- en [2] : la réponse du service web. Elle est différente de la précédente qui était [401 Unauthorized]. Cette fois-ci, l'utilisateur s'est authentifié correctement mais n'a pas les droits suffisants pour accéder à l'URL ;

2.15 Conclusion

Rapelons l'architecture globale de notre application client / serveur :



Un service web sécurisé est maintenant opérationnel. On verra qu'il devra être modifié suite à des problèmes qui vont se révéler à la construction du client Angular JS. Mais nous attendrons de rencontrer le problème pour le résoudre. Nous allons maintenant construire le client Angular qui va offrir une interface web pour gérer les rendez-vous des médecins.

3 Le client Angular JS

3.1 Références du framework Angular JS

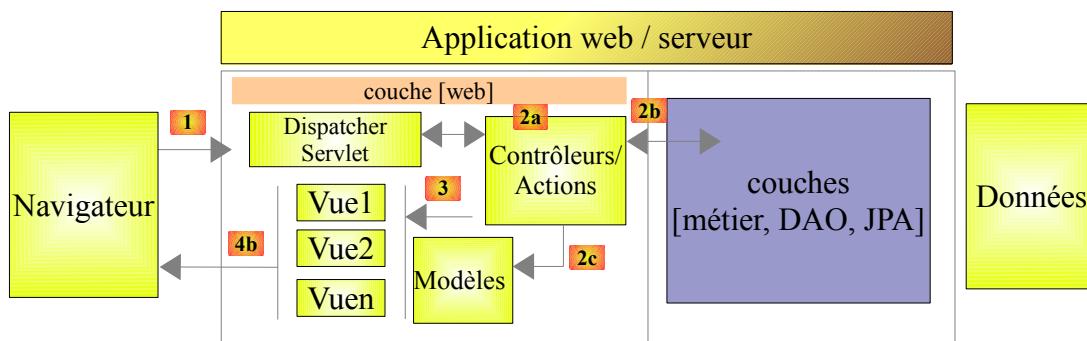
Deux références ont été données pour le framework Angular JS au début de ce document. Nous les redonnons ici :

- [ref1] : le livre " **Pro AngularJS**" écrit par **Adam Freeman** aux éditions **Apress**. C'est un excellent livre. Les codes source des exemples de ce livre sont disponibles gratuitement à l'URL [\[http://www.apress.com/downloadable/download/sample/sample_id/1527/\]](http://www.apress.com/downloadable/download/sample/sample_id/1527/);
- [ref2] : la documentation officielle d'Angular JS [<https://docs.angularjs.org/guide>];

Angular JS mérite un livre à lui tout seul. Celui d'Adam Freeman a plus de 600 pages et elles ne sont pas gaspillées. Nous allons décrire une application Angular et au cours de cette description nous serons amenés à parler des fondamentaux de ce framework. Néanmoins, nous nous en tiendrons aux seules explications nécessaires à la compréhension de la solution proposée. Angular est un framework extrêmement riche et il existe de nombreuses solutions pour arriver au même résultat. C'est une difficulté car lorsqu'on débute, on ne sait pas si on utilise une solution moins bonne ou meilleure qu'une autre. C'est le cas de la solution proposée ici. Elle pourrait être écrite différemment et peut-être avec de meilleures pratiques.

3.2 Architecture du client Angular

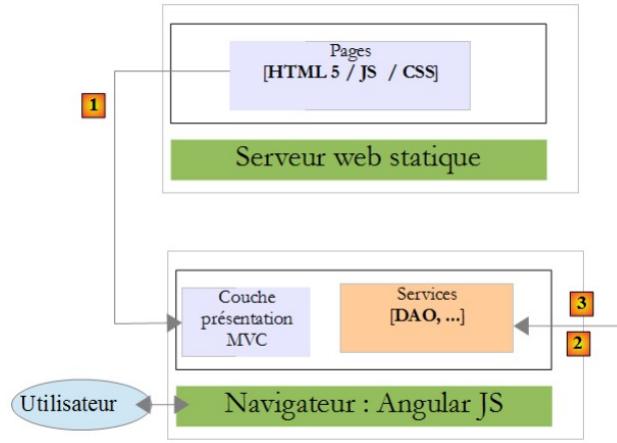
L'architecture du client Angular ressemble à celle d'une application web MVC classique avec quelques différences. Une application web Spring MVC a par exemple l'architecture suivante :



Le traitement d'une demande d'un client se déroule de la façon suivante :

1. **demande** - les URL demandées sont de la forme `http://machine:port/contexte/Action/param1/param2/...?p1=v1&p2=v2...`. La [Dispatcher Servlet] est la classe de Spring qui traite les URL entrantes. Elle "route" l'URL vers l'action qui doit la traiter. Ces actions sont des méthodes de classes particulières appelées [Contrôleurs]. Le **C** de MVC est ici la chaîne [Dispatcher Servlet, Contrôleur, Action]. Si aucune action n'a été configurée pour traiter l'URL entrante, la servlet [Dispatcher Servlet] répondra que l'URL demandée n'a pas été trouvée (erreur 404 NOT FOUND) ;
2. **traitement**
 - l'action choisie peut exploiter les paramètres *param_i* que la servlet [Dispatcher Servlet] lui a transmis. Ceux-ci peuvent provenir de plusieurs sources :
 - du chemin [/param1/param2/] de l'URL,
 - des paramètres [p1=v1&p2=v2] de l'URL,
 - de paramètres postés par le navigateur avec sa demande ;
 - dans le traitement de la demande de l'utilisateur, l'action peut avoir besoin de la couche [metier] [2b]. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
 - une page d'erreur si la demande n'a pu être traitée correctement
 - une page de confirmation sinon
 - l'action demande à une certaine vue de s'afficher [3]. Cette vue va afficher des données qu'on appelle le **modèle de la vue**. C'est le **M** de MVC. L'action va créer ce modèle M [2c] et demander à une vue V de s'afficher [3] ;
3. **réponse** - la vue V choisie utilise le modèle M construit par l'action pour initialiser les parties dynamiques de la réponse HTML qu'elle doit envoyer au client puis envoie cette réponse.

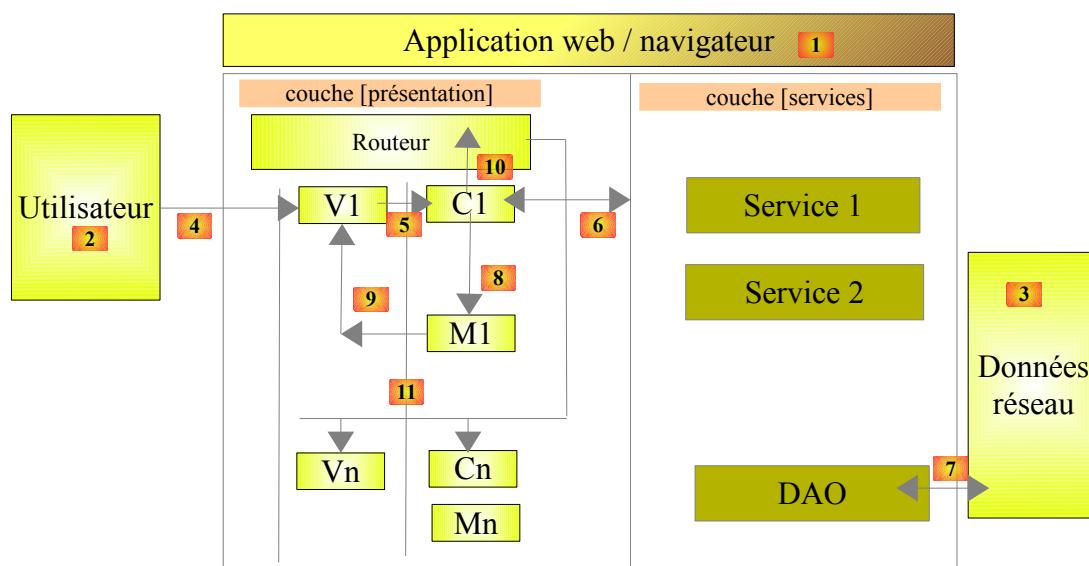
L'architecture de notre client Angular sera analogue avec une terminologie un peu différente. Tout d'abord les applications Angular sont généralement des applications web à page unique (APU) ou Single Page Application (SPA) :



- l'utilisateur demande l'URL initiale de l'application sous la forme : `http://machine:port/contexte`. Le navigateur va interroger un serveur web pour obtenir le document demandé. Celui-ci est une page HTML stylisée par du CSS et rendue dynamique par du Javascript ;
- ensuite l'utilisateur va interagir avec les vues qui lui sont présentées. On peut distinguer diverses sortes d'interactions :
 - celles qui ne nécessitent aucune interaction avec l'extérieur, par exemple cacher / montrer des éléments de la vue. Elle sont traitées par le Javascript embarqué ;
 - celles qui nécessitent des données provenant d'un service web distant. Elles vont être récupérées par un appel AJAX (Asynchronous Javascript And Xml), un modèle va être construit et une vue affichée ;
 - celles qui nécessitent une autre vue que la vue initiale. Elle va être demandée par un appel Ajax au serveur qui a délivré la page initiale. Puis le processus précédent va se répéter. La page obtenue va être mise en cache dans le navigateur. Au prochain appel, elle ne sera pas demandée au serveur HTML distant ;

Au final, le navigateur ne fait qu'un seul appel HTTP, celui qui obtient la page initiale. Les appels HTTP suivants, vers le serveur de pages HTML ou des services web distants, sont faits par le Javascript embarqué dans les pages.

Nous présentons maintenant l'architecture de l'application au sein du navigateur. Nous oublions le serveur HTML qui délivre les pages HTML de l'application. Pour l'explication, on peut considérer qu'elles sont toutes présentes au sein du cache du navigateur.



Tout d'abord, il faut situer cette architecture :

- en [1], on est dans un navigateur ;
- en [2], un utilisateur interagit avec les vues affichées par celui-ci ;
- en [3], les données sont cherchées sur le réseau, souvent auprès de services web ;

L'utilisateur interagit avec des vues : il remplit des formulaires et les valide. Explicitons ce processus avec la vue **V1** ci-dessus. On supposera que c'est la vue initiale de l'application. Elle a été obtenue de la façon suivante :

- l'utilisateur demande l'URL initiale de l'application sous la forme : `http://machine:port/contexte` ;
- le navigateur a demandé le document associé à cette URL. Il a reçu la page HTML / CSS / JS de la vue **V1** ;
- le Javascript embarqué dans la page a alors pris la main et a donné le contrôle au contrôleur **C1** [5] ;
- celui-ci a construit le modèle **M1** [8] [9] de la vue **V1**. La construction de ce modèle a pu nécessiter l'utilisation de services internes [6] et l'interrogation de services externes [7] ;

L'utilisateur a maintenant une vue **V1** devant lui. Imaginons que c'est un formulaire. Il le remplit puis le valide :

- en [4], l'utilisateur valide le formulaire ;
- en [5], cet événement va être traité par l'une des méthodes du contrôleur **C1** ;

Si l'événement n'entraîne qu'un simple changement de la vue **V1** (cacher / montrer des zones), le contrôleur **C1** va modifier le modèle **M1** de la vue **V1** puis afficher de nouveau la vue **V1**. Il peut pour ce faire avoir besoin de l'un des services de la couche [services] [6].

Si l'événement nécessite des données externes :

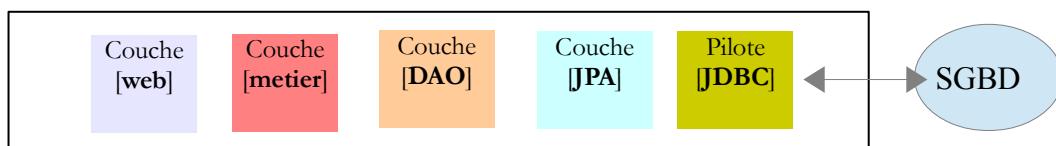
- en [6], le contrôleur **C1** va demander à la couche [DAO] de les obtenir ;
- en [7], celle-ci va faire un ou plusieurs appels AJAX pour les obtenir ;
- en [8] et [9], le modèle **M1** va être modifié et la vue **V1** affichée ;

Si l'événement entraîne un changement de vue, dans les deux cas précédents, au lieu d'afficher la vue **V1**, le contrôleur **C1** va demander une nouvelle URL [10]. C'est une URL interne au navigateur. Elle ne se traduit pas immédiatement par un appel HTTP au serveur de pages HTML. Ce changement d'URL est traité par un routeur configuré de telle sorte qu'à chaque URL interne correspond une vue V et son contrôleur C. Le routeur fait alors afficher la nouvelle vue **Vn**. Avant l'affichage, son contrôleur **Cn** prend la main, construit le modèle **Mn** puis fait afficher la vue **Vn** [11]. Si la page HTML de la vue **Vn** n'était pas en cache dans le navigateur, alors elle sera demandée au serveur de pages HTML.

La couche [Présentation] de cette architecture est proche de l'architecture JSF (Java Server Faces) :

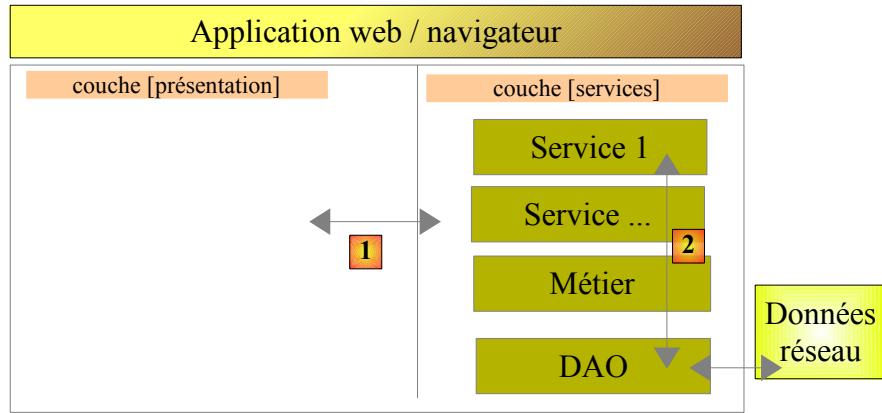
- la vue **V** correspond à la vue de type Facelet de JSF ;
- le contrôleur **C** correspond au bean JSF, une classe Java qui contient à la fois le modèle **M** de la vue **V** et les gestionnaires des événements de celle-ci ;

La couche [Services] est différente des couches [Services] auxquelles on est habitué. En développement web, côté serveur, on a le plus souvent l'architecture en couches suivante :



Ci-dessus, la couche [web] ne communique avec la couche [DAO] qu'au travers de la couche [métier]. Rien ne nous empêcherait d'injecter dans la couche [web] une référence sur la couche [DAO] qui permettrait cette communication. Mais on se l'interdit.

Avec Angular, on ne se l'interdit pas. L'architecture devient alors la suivante :



- en [1], la couche [présentation] peut communiquer directement avec n'importe quel service ;
- en [2], les services se connaissent entre-eux. Un service peut en utiliser un ou plusieurs autres.

3.3 Les vues du client Angular

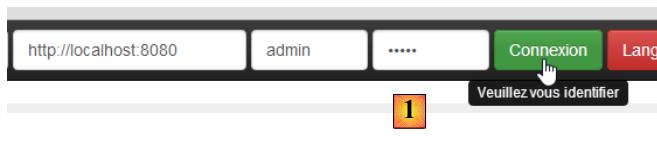
Les vues du client Angular ont déjà été présentées au paragraphe 1.3.3, page 11. Pour faciliter la lecture de ce nouveau chapitre, nous les redonnons ici. La première vue est la suivante :

The screenshot shows a web browser window with the title "RdvMedecins". The address bar shows "cins-angular/app/app.html#/". The page itself has a header with a logo (a caduceus) and the text "Cabinet Médical Les Médecins Associés". Below the header is a login form. The form includes fields for "username" (with value "admin") [11], "password" (with value "admin") [12], and a "Connexion" button [13]. There is also a "Langue" dropdown menu [14]. At the top of the page, there is a toolbar with a "mode debug" checkbox [7], a "waitingTimeBeforeTask" input field [8] containing "0", a "duration" input field [9] containing "9", and a URL input field [10] containing "http://localhost:8080". A tooltip at the bottom left provides information about the "waitingTimeBeforeTask" model.

```
Modèle : { "waitingTimeBeforeTask": 0, "delay": { "on": true }, "titre": { "text": "identification", "show": true, "model": {} }, "navbarrun": { "show": false }, "navbarstart": { "show": true }, "errors": { "show": false }, "view": { "url": "/", "model": {}, "done": false }, "waiting": { "text": "msg_waiting_init", "show": false }, "task": { "action": { "promise": {} }, "isFinished": false }, "serverUrl": "http://localhost:8080", "username": "admin", "password": "admin" }
```

- en [6], la page d'entrée de l'application. Il s'agit d'une application de prise de rendez-vous pour des médecins ;
- en [7], une case à cocher qui permet d'être ou non en mode [debug]. Ce dernier se caractérise par la présence du cadre [8] qui affiche le modèle de la vue courante ;
- en [9], une durée d'attente artificielle en millisecondes. Elle vaut 0 par défaut (pas d'attente). Si N est la valeur de ce temps d'attente, toute action de l'utilisateur sera exécutée après un temps d'attente de N millisecondes. Cela permet de voir la gestion de l'attente mise en place par l'application ;
- en [10], l'URL du serveur Spring 4. Si on suit ce qui a précédé, c'est [http://localhost:8080];
- en [11] et [12], l'identifiant et le mot de passe de celui qui veut utiliser l'application. Il y a deux utilisateurs : **admin/admin** (login/password) avec un rôle (ADMIN) et **user/user** avec un rôle (USER). Seul le rôle ADMIN a le droit d'utiliser l'application. Le rôle USER n'est là que pour montrer ce que répond le serveur dans ce cas d'utilisation ;

- en [13], le bouton qui permet de se connecter au serveur ;
- en [14], la langue de l'application. Il y en a deux : le français par défaut et l'anglais.



- en [1], on se connecte ;

The screenshot shows the main application interface. At the top, the title 'Cabinet Médical Les Médecins' is displayed. To the left, there is a logo of a caduceus. Above the logo, the text 'RdvMedecins' is followed by a dropdown menu with 'Agenda' selected, which has a tooltip 'Afficher l'agenda du médecin choisi pour le jour choisi'. Below the logo is a message: 'Choisissez un médecin et un jour pour avoir l'agenda'. On the left, there is a dropdown menu for 'Médecin' with 'Mme Marie PELISSIER' selected. On the right, there is a date picker for 'Jour' showing the month of June 2014. The date '05' is highlighted in blue, indicating it is the selected day.

- une fois connecté, on peut choisir le médecin avec lequel on veut un rendez-vous [2] et le jour de celui-ci [3] ;
- on demande en [4] à voir l'agenda du médecin choisi pour le jour choisi ;



Cabinet Médical Les Médecins

Rendez - vous de Mme Marie PELISSIER le 2014-06-05

Créneau horaire	Client	Action
08h00:08h20		Réserver  
08h20:08h40		Réserver 
08h40:09h00		Réserver
09h00:09h20		Réserver

- une fois obtenu l'agenda du médecin, on peut réserver un créneau [5] ;



Cabinet Médical Les Médecins Ass

Rendez - vous de Mme Marie PELISSIER le 2014-06-05 à 08h00:08h20

Client 

Mr Jules MARTIN 

- en [6], on choisit le patient pour le rendez-vous et on valide ce choix en [7] ;



Cabinet Médical Les Médecins Associés

Rendez - vous de Mme Marie PELISSIER le 2014-06-05

Créneau horaire	Client	Action
08h00:08h20	Mr Jules MARTIN	Supprimer 7
08h20:08h40		Réserver
08h40:09h00		Réserver
09h00:09h20		Réserver

Une fois le rendez-vous validé, on est ramené automatiquement à l'agenda où le nouveau rendez-vous est désormais inscrit. Ce rendez-vous pourra être ultérieurement supprimé [7].

Les principales fonctionnalités ont été décrites. Elles sont simples. Celles qui n'ont pas été décrites sont des fonctions de navigation pour revenir à une vue précédente. Terminons par la gestion de la langue :

RdvMedecins Agenda mode debug 0 Déconnexion Langue

French English 1

Cabinet Médical Les Médecins Associés

Choisissez un médecin et un jour pour avoir l'agenda

Médecin

Mme Marie PELISSIER ▾

Jour

juin 2014						
dim.	lun.	mar.	mer.	jeu.	ven.	sam.
22	01	02	03	04	05	06
23	08	09	10	11	12	13
24	15	16	17	18	19	20
25	22	23	24	25	26	27
26	29	30	01	02	03	04
27	06	07	08	09	10	11
						12

- en [1], on passe du français à l'anglais ;

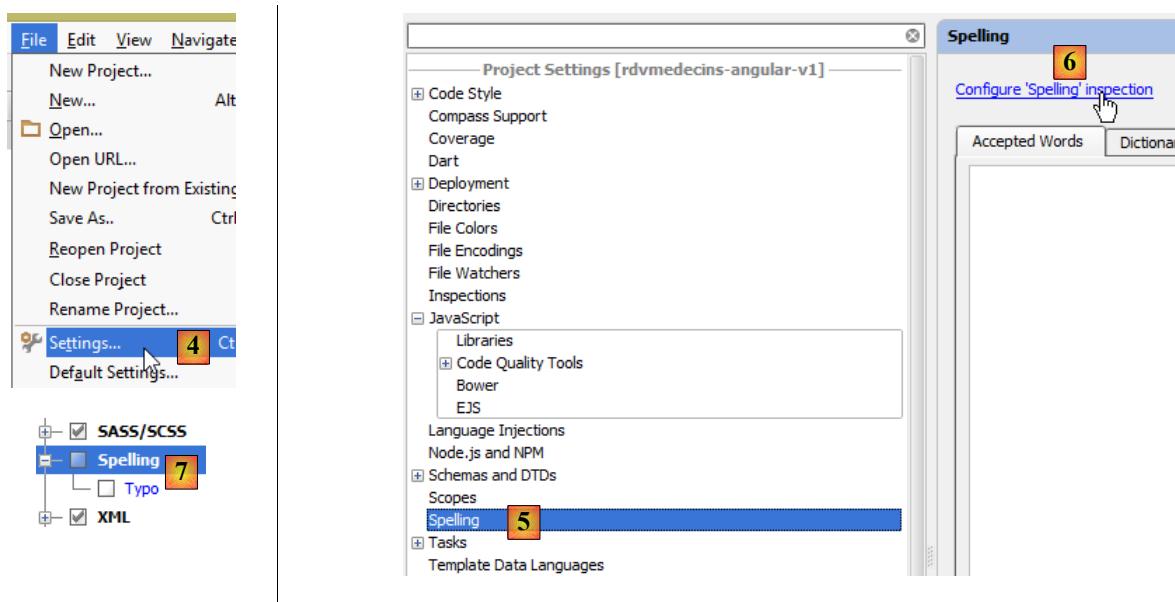
- en [2], la vue est passée en anglais, y-compris le calendrier ;

3.4 Configuration du projet Angular

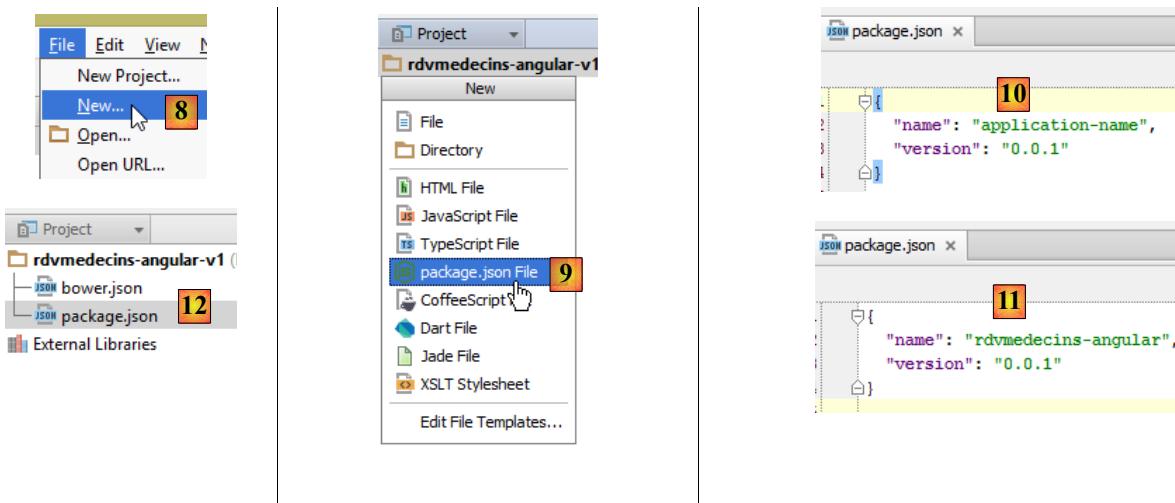
Nous allons construire notre client Angular de façon progressive. Nous utilisons l'IDE [Webstorm](#).

Créons un dossier vide [rdvmedecins-angular-v1] puis ouvrons-le avec Webstorm :

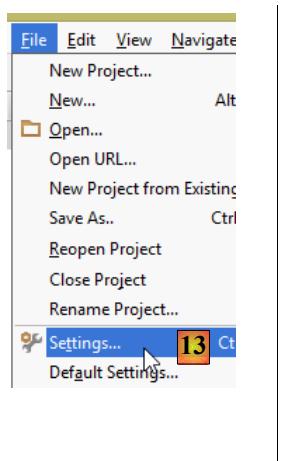
- en [1], on ouvre un dossier ;
- en [2], on désigne le dossier que nous avons créé ;
- en [3], nous obtenons un projet Webstorm vide ;



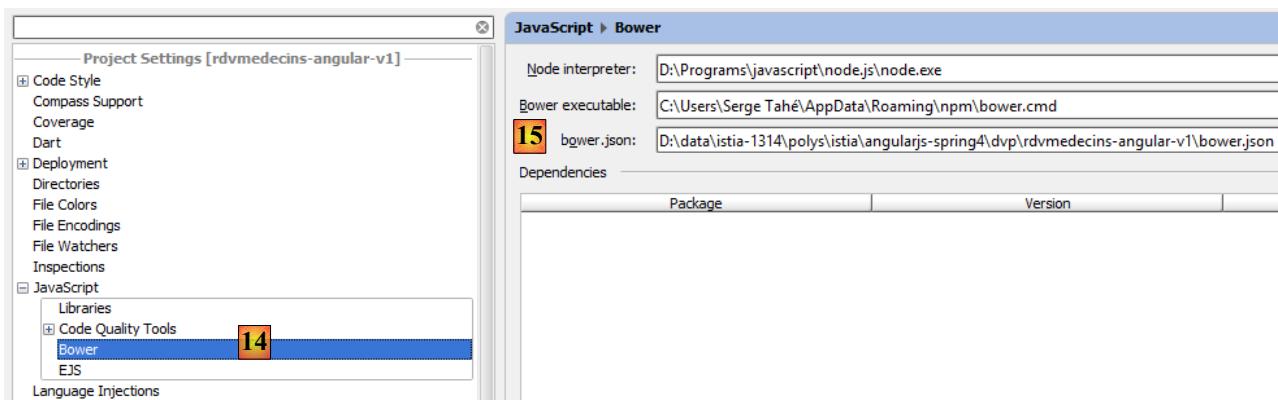
- en [4], la configuration du projet se fait par l'option [File / Settings] ;
- en [5] et [6], on configure la propriété [Spelling] qui gère la vérification orthographique. Par défaut, celle-ci est active. Comme le logiciel téléchargé est en langue anglaise, nos commentaires en français des programmes vont être soulignés comme de possibles erreurs d'orthographe. On désactive donc cette vérification orthographique [7] ;



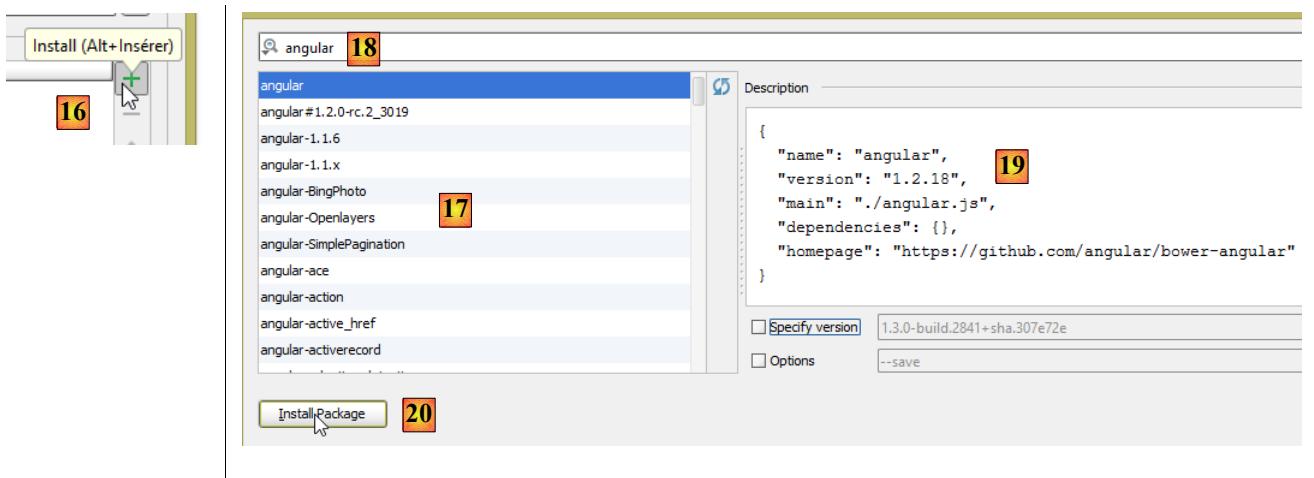
- en [8], on crée un nouveau fichier ;
- en [9], on choisit de créer le fichier [package.json] qui décrit l'application avec une syntaxe JSON ;
- en [10], le fichier généré qu'on modifie comme montré en [11] ;
- en [12], on sauvegarde ce fichier à la fois dans [package.json] et [bower.json] ;



- en [13], on configure de nouveau le projet ;

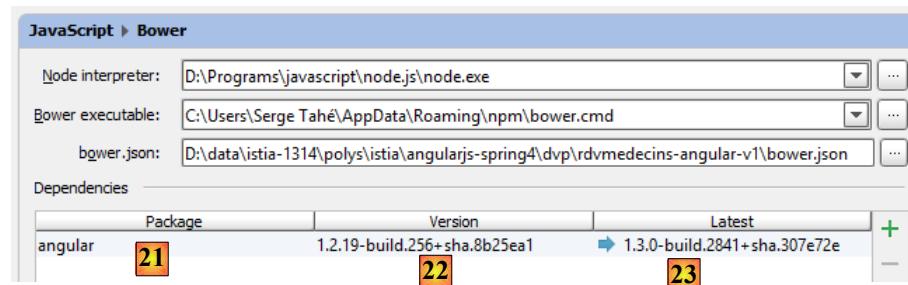


- en [14], on configure la propriété [Javascript / Bower] qui va nous permettre de déclarer les bibliothèques Javascript dont nous avons besoin ;
- en [15], désigner le fichier [bower.json] que nous venons de créer ;



- en [16], ajoutons une bibliothèque Javascript ;
- en [17] sont affichées toutes les bibliothèques Javascript téléchargeables ;
- en [18], nous pouvons mettre un terme pour filtrer la liste [17]. Ici, nous indiquons que nous voulons la bibliothèque [Angular JS] ;
- en [19], apparaissent les caractéristiques de la bibliothèque. On voit ici que la version 1.2.18 d'Angular va être téléchargée ;

- en [20], on la télécharge ;

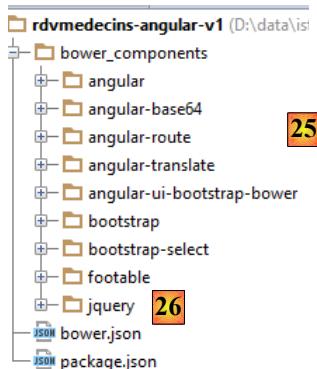


- en [21], on voit qu'elle a été téléchargée ;
- en [22], on voit la version téléchargée. C'est donc en réalité la 1.2.19 ;
- en [23], on voit la dernière version disponible ;

Package	Version	Latest
angular	1.2.19-build.256+sha.8b25ea1	1.3.0-build.2841+sha.307e72e
angular-base64	2.0.2	2.0.2
angular-i18n	1.2.19-build.256+sha.8b25ea1	1.3.0-build.2841+sha.307e72e
angular-route	1.2.19-build.256+sha.8b25ea1	1.3.0-build.2841+sha.307e72e
angular-translate	2.2.0	2.2.0
angular-ui-bootstrap-bower	0.11.0	0.11.0
bootstrap	3.1.1	3.1.1
bootstrap-select	1.5.2	1.5.2
footable	2.0.1	2.0.1

- en [24], en suivant la même démarche que précédemment, on télécharge les bibliothèques suivantes :

angular-base64	pour encoder la chaîne " user:password " en Base64 ;	https://github.com/ninjatronic/angular-base64
angular-i18n	pour internationaliser le calendrier	https://code.angularjs.org/1.2.18/i18n/
angular-route	pour router les URL internes de l'application vers le bon contrôleur et la bonne vue ;	https://docs.angularjs.org/api/ngRoute
angular-translate	permet l'internationalisation des vues. C'est un projet indépendant d'Angular. Ici, deux langues seront utilisées : le français et l'anglais ;	https://github.com/angular-translate/angular-translate
angular-ui-bootstrap-bower	fournit des composants visuels compatibles Bootstrap. On utilisera ici son calendrier ;	https://github.com/angular-ui/bootstrap
bootstrap	le framework CSS Bootstrap. Sera utilisé pour construire les vues ;	http://getbootstrap.com/
footable	fournit un composant visuel de type " tableau ". Il est " responsive " en ce sens qu'il peut s'adapter à la taille de l'écran ;	http://themergency.com/footable/
bootstrap-select	fournit un composant de type " liste déroulante " ;	http://silviomoreto.github.io/bootstrap-select/



- en [25], les bibliothèques téléchargées ont été installées dans le dossier [bower_components] ;
- en [26], on voit que la bibliothèque JQuery a été téléchargée. C'est parce que Bootstrap l'utilise. Le système d'installation des dépendances Javascript d'un projet est analogue à celui de Maven pour le monde Java : si une bibliothèque téléchargée a elle-même des dépendances, celles-ci sont automatiquement téléchargées ;

Le fichier [bower.json] a évolué :

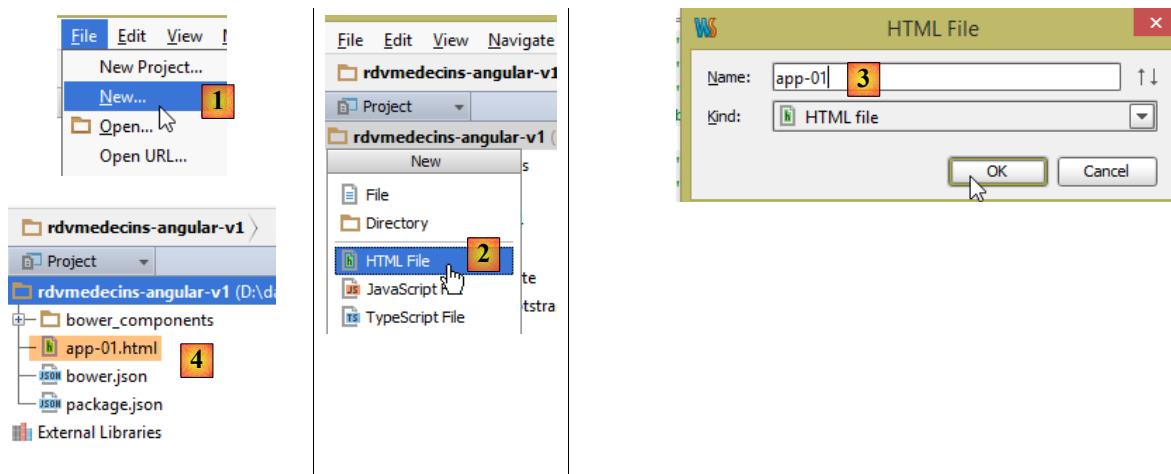
```

1.  {
2.    "name": "rdvmedecins-angular",
3.    "version": "0.0.1",
4.    "dependencies": {
5.      "angular": "~1.2.18",
6.      "angular-base64": "~2.0.2",
7.      "angular-route": "~1.2.18",
8.      "angular-translate": "~2.2.0",
9.      "bootstrap": "~3.1.1",
10.     "footable": "~2.0.1",
11.     "angular-ui-bootstrap-bower": "~0.11.0",
12.     "bootstrap-select": "~1.5.2"
13.   }
14. }
```

Toutes les dépendances téléchargées ont été inscrites dans le fichier.

3.5 La page initiale du client Angular

Nous créons une première version de la page initiale du client Angular :



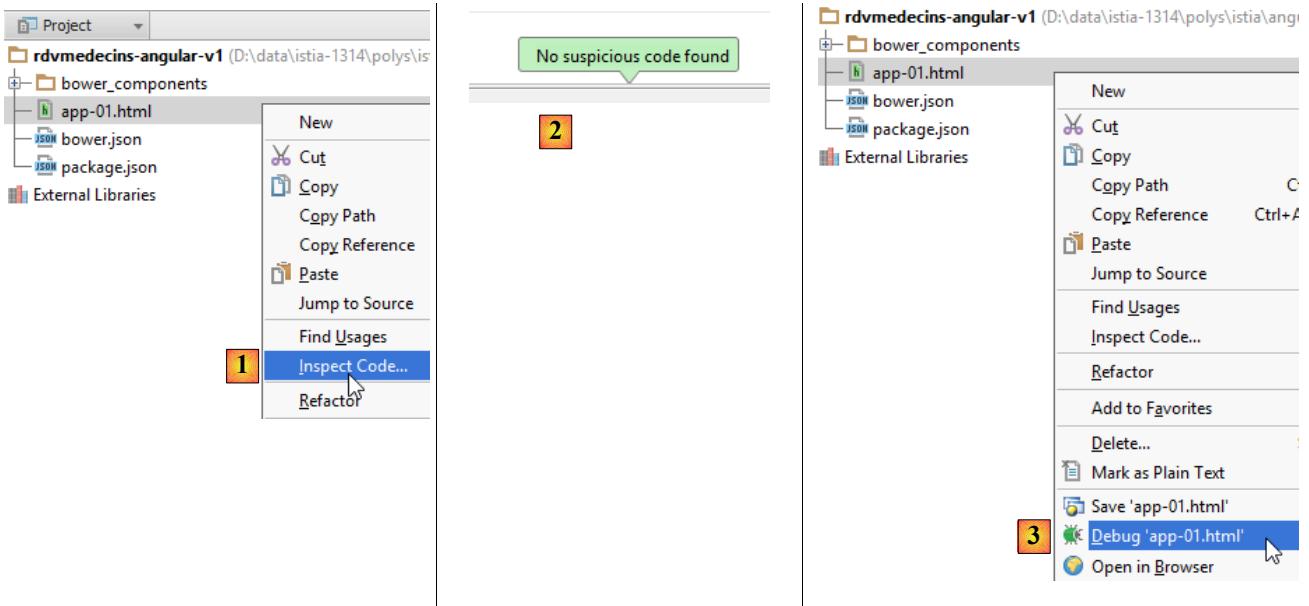
- en [1] et [2], nous créons un fichier HTML nommé [app-01] [3] et [4] ;

Le fichier [app-01.html] va être notre page principale pendant un moment. Nous allons y configurer l'importation des fichiers CSS et JS dont l'application a besoin :

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>RdvMedecins</title>
5.   <!-- META -->
6.   <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
7.   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8.   <meta name="description" content="Angular client for RdvMedecins">
9.   <meta name="author" content="Serge Tahé">
10.  <!-- le CSS -->
11.  <link href="bower_components/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet" />
12.  <link href="bower_components/bootstrap/dist/css/bootstrap-theme.min.css" rel="stylesheet"/>
13.  <link href="bower_components/bootstrap-select/bootstrap-select.min.css" rel="stylesheet"/>
14.  <link href="bower_components/footable/css/footable.core.min.css" rel="stylesheet"/>
15. </head>
16. <body>
17. <div class="container">
18.   <h1>Rdvmedecins - v1</h1>
19. </div>
20. <!-- Bootstrap core JavaScript ===== -->
21. <script type="text/javascript" src="bower_components/jquery/dist/jquery.min.js"></script>
22. <script type="text/javascript" src="bower_components/bootstrap/dist/js/bootstrap.min.js"></script>
23. <script type="text/javascript" src="bower_components/bootstrap-select/bootstrap-select.min.js"></script>
24. <script type="text/javascript" src="bower_components/footable/dist/footable.min.js"></script>
25. <!-- angular js -->
26. <script type="text/javascript" src="bower_components/angular/angular.min.js"></script>
27. <script type="text/javascript" src="bower_components/angular-ui-bootstrap/ui-bootstrap-tpls.min.js"></script>
28. <script type="text/javascript" src="bower_components/angular-route/angular-route.min.js"></script>
29. <script type="text/javascript" src="bower_components/angular-translate/angular-translate.min.js"></script>
30. <script type="text/javascript" src="bower_components/angular-base64/angular-base64.min.js"></script>
31. </body>
32. </html>
```

- lignes 11-12 : les fichiers CSS pour Bootstrap ;
- ligne 13 : le fichier CSS pour le composant [bootstrap-select] ;
- ligne 14 : le fichier CSS pour le composant [footable] ;
- lignes 21-24 : les fichiers JS des composants Bootstrap ;
- ligne 21 : les composants Bootstrap sont propulsés par JQuery ;
- ligne 22 : le fichier JS de Bootstrap ;
- ligne 23 : le fichier JS pour le composant [bootstrap-select] ;
- ligne 24 : le fichier JS pour le composant [footable] ;
- lignes 26-30 : les fichiers JS d'Angular et des projets qui s'y raccrochent ;
- ligne 26 : le fichier JS d'Angular. Il doit être chargé après JQuery si cette bibliothèque est utilisée ;
- ligne 27 : le fichier JS du projet [angular-ui-bootstrap] ;
- ligne 28 : le fichier JS du routeur [angular-route] ;
- ligne 29 : le fichier JS du module d'internationalisation des applications Angular ;
- ligne 30 : le fichier JS du module [angular-base64] ;

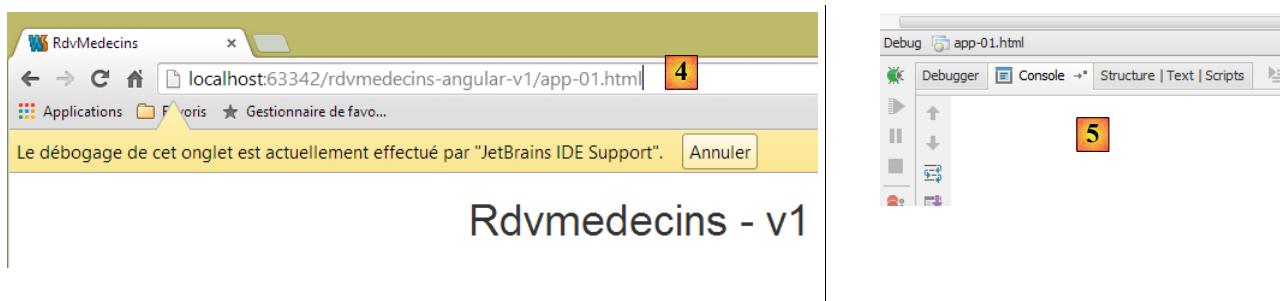
La validité du fichier [app-01.html] peut être vérifiée :



- en [1], on demande l'inspection du code ;
- en [2], le résultat lorsque tout va bien ;

Cette inspection systématique du code avant son exécution est conseillée. Ici, cette détection permet de détecter toute erreur de référence des fichiers CSS et JS. Si un chemin est incorrect, l'inspecteur de code le signalera.

- en [3], la page peut être chargée dans un navigateur par un débogueur. On obtient le résultat suivant dans le navigateur :

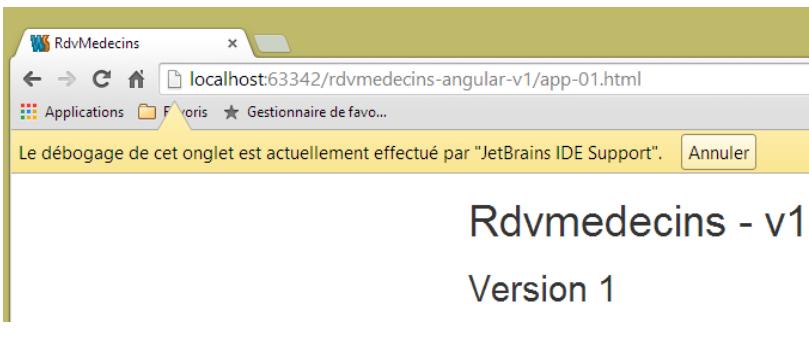


- en [4], la page [app-01.html] a été délivrée par un serveur interne à Webstorm opérant ici sur le port 63342 ;
- en [5], la console du débogueur. Si des erreurs s'étaient produites, elles seraient apparues ici. C'est également là que vont les affichages écran produits par l'instruction [console.log(expression)] du Javascript. Nous utiliserons abondamment cette possibilité ;

Le mode débogage permet de modifier la page dans Webstorm et de voir les résultats de ces modifications dans le navigateur sans avoir à recharger la page. Ainsi si nous ajoutons la ligne 3 ci-dessous :

```
1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.   <h2>Version 1</h2>
4. </div>
```

et que nous revenons au navigateur, nous constatons que la page a changé :



3.6 Découverte de Bootstrap

Nous allons illustrer maintenant certaines des caractéristiques de Bootstrap utilisées dans l'application. Je n'ai qu'une connaissance limitée de ce framework, obtenue par des copier / coller de codes trouvés sur Internet. J'expliquerai le rôle des classes CSS que je crois comprendre. Je m'abstiendrai de commenter les autres.

3.6.1 Exemple 1

Dans Angular, les opérations qui vont chercher de l'information à l'extérieur sont **asynchrones**. Cela signifie que l'opération est lancée et qu'il y a retour immédiat à la vue avec laquelle l'utilisateur peut continuer à interagir. L'application est avertie de la fin de l'opération par un événement. Cet événement est traité par une fonction JS qui peut alors enrichir la vue actuelle ou en changer. Si l'opération est susceptible d'être longue, il est utile d'offrir à l'utilisateur la possibilité de l'annuler. Nous la lui offrirons systématiquement. Pour cela, nous utiliserons un bandeau Bootstrap :



Pour obtenir ce résultat, nous dupliquons [app-01.html] dans [app-02.html] et nous modifions les lignes suivantes :

```
1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.   <div class="alert alert-warning">
4.     <h1>Opération en cours. Veuillez patienter...
5.     <button class="btn btn-primary pull-right">Annuler</button>
6.     
7.   </h1>
8. </div>
9. </div>
```

- ligne 1 : la classe CSS [container] définit une zone d'affichage à l'intérieur du navigateur ;
- ligne 3 : la classe CSS [alert] affiche une zone colorée. La classe [alert-warning] utilise une couleur prédéfinie ;
- ligne 5 : la classe [btn] habille un bouton. La classe [btn-primary] lui donne une certaine couleur. La classe [pull-right] l'envoie sur la droite du bandeau d'alerte ;
- ligne 6 : une image animée d'attente ;

3.6.2 Exemple 2

Les différentes vues de l'application auront un titre commun :



Pour obtenir ce résultat, nous dupliquons [app-01.html] dans [app-03.html] et nous modifions les lignes suivantes :

```

1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.   <!-- Bootstrap Jumbotron -->
4.   <div class="jumbotron">
5.     <div class="row">
6.       <div class="col-md-2">
7.         
8.       </div>
9.       <div class="col-md-10">
10.        <h1>Les Médecins associés</h1>
11.      </div>
12.    </div>
13.  </div>
14. </div>
```

- la zone colorée est obtenue avec la classe [jumbotron] de la ligne 4 ;
- ligne 5 : la classe [row] définit une ligne à 12 colonnes ;
- ligne 6 : la classe [col-md-2] définit une zone de deux colonnes dans la ligne ;
- ligne 7 : dans ces deux colonnes on met une image ;
- lignes 9-11 : dans les 10 autres colonnes, on met le texte ;

3.6.3 Exemple 3

Les vues auront un bandeau haut de commande. On y trouvera des options de commande, liens ou boutons. On y trouvera également des éléments de formulaire. Par exemple :



Pour obtenir ce résultat, nous dupliquons [app-01.html] dans [app-04.html] et nous modifions les lignes suivantes :

```

1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
5.     <div class="container">
6.       <div class="navbar-header">
7.         <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
8.           <span class="sr-only">Toggle navigation</span>
9.           <span class="icon-bar"></span>
```

```

10.      <span class="icon-bar"></span>
11.      <span class="icon-bar"></span>
12.    </button>
13.    <a class="navbar-brand" href="#">RdvMedecins</a>
14.  </div>
15.  <div class="navbar-collapse collapse">
16.    <form class="navbar-form navbar-right">
17.      <!-- mode debug -->
18.      <label style="width: 100px">
19.        <input type="checkbox">
20.        <span style="color: white">Debug</span>
21.      </label>
22.      <!-- formulaire d'identification -->
23.      <div class="form-group">
24.        <input type="text" class="form-control" placeholder="Temps d'attente"
25.               style="width: 150px"/>
26.        <input type="text" class="form-control" placeholder="URL du service web"
27.               style="width: 200px"/>
28.        <input type="text" class="form-control" placeholder="Login"
29.               style="width: 100px"/>
30.        <input type="password" class="form-control" placeholder="Mot de passe"
31.               style="width: 100px"/>
32.      </div>
33.      <button class="btn btn-success">
34.        Connexion
35.      </button>
36.    </form>
37.  </div>
38.  <button class="btn btn-success">
39.    Connexion
40.  </button>
41. </form>
42. </div>
43. </div>
44. </div>
45. </div>

```

- ligne 4 : la classe [navbar] va styler la barre de navigation. La classe [navbar-inverse] lui donne le fond noir. La classe [navbar-fixed-top] va faire en sorte que lorsqu'on 'scrolle' la page affichée par le navigateur, la barre de navigation va rester en haut de l'écran ;
- lignes 6-14 : définissent la zone [1]. C'est typiquement une série de classes que je ne comprends pas. J'utilise le composant tel quel ;
- ligne 15 : définissent une zone 'responsive' de la barre de commande. Sur un smartphone, cette zone disparaît dans une zone de menu ;
- ligne 16 : la classe [navbar-form] habille un formulaire de la barre de commande. La classe [navbar-right] le rejette à droite de celle-ci ;
- lignes 23-32 : les quatre zones de saisie du formulaire de la ligne 17 [3]. Elles sont à l'intérieur d'une classe [form-group] qui habille les éléments d'un formulaire et chacune d'elles a la classe [form-control] ;
- ligne 33 : la classe [btn] qu'on a déjà rencontrée, enrichie de la classe [btn-success] qui lui donne sa couleur verte ;

3.6.4 Exemple 4

Le bandeau de commande permettra de changer de langue grâce à une liste déroulante :



Pour obtenir ce résultat, nous dupliquons [app-01.html] dans [app-05.html] et nous ajoutons les ligne suivantes à la barre de commande :

```
1.      <button class="btn btn-success">
```

```
2.         Connexion
3.     </button>
4.     <!-- langues --&gt;
5.     &lt;div class="btn-group"&gt;
6.         &lt;button type="button" class="btn btn-danger"&gt;
7.             Langues
8.         &lt;/button&gt;
9.         &lt;button type="button" class="btn btn-danger dropdown-toggle" data-toggle="dropdown"&gt;
10.            &lt;span class="caret"&gt;&lt;/span&gt;
11.            &lt;span class="sr-only"&gt;Toggle Dropdown&lt;/span&gt;
12.        &lt;/button&gt;
13.        &lt;ul class="dropdown-menu" role="menu"&gt;
14.            &lt;li&gt;
15.                &lt;a href=""&gt;Français&lt;/a&gt;
16.            &lt;/li&gt;
17.            &lt;li&gt;
18.                &lt;a href=""&gt;English&lt;/a&gt;
19.            &lt;/li&gt;
20.        &lt;/ul&gt;
21.    &lt;/div&gt;
22. &lt;/form&gt;</pre>
```

Les lignes rajoutées sont les lignes 4-21.

- ligne 5 : la classe [btn-group] habille un groupe de boutons. Il y en a deux aux lignes 6 et 9 ;
 - lignes 6-8 : le premier bouton définit le libellé de la liste déroulante. La classe [btn-danger] lui donne sa couleur rouge ;
 - lignes 9-12 : le second bouton est celui de la liste déroulante. Il est accolé au premier, ce qui donne l'impression d'un composant unique ;
 - ligne 10 : affiche la flèche descendante indiquant que le bouton est une liste déroulante ;
 - ligne 11 : pour des 'screen readers' ;
 - lignes 13-20 : les éléments de la liste déroulante sont les éléments d'une liste non ordonnée ;

3.6.5 Exemple 5

Pour valider un formulaire ou pour naviguer, l'utilisateur disposera dans la barre de commande d'options ou de boutons comme ci-dessous :



Des options de menu ont été installées en [1]. Pour obtenir ce résultat, nous dupliquons [app-01.html] dans [app-06.html] et nous ajoutons les ligne suivantes :

```
1. <div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
2.   <div class="container">
3.     <div class="navbar-header">
4.     ...
5.       </div>
6.       <!-- options de menu -->
7.       <div class="collapse navbar-collapse">
8.         <ul class="nav navbar-nav">
9.           <li class="active">
10.             <a href="">
11.               <span>Home</span>
12.             </a>
13.           </li>
14.           <li class="active">
15.             <a href="">
16.               <span>Agenda</span>
17.             </a>
18.           </li>
19.           <li class="active">
20.             <a href="">
21.               <span>Valider</span>
```

```

22.          </a>
23.      </li>
24.      <li class="active">
25.          <a href="">
26.              <span>Annuler</span>
27.          </a>
28.      </li>
29.  </ul>
30.  <!-- boutons de droite -->
31.  <form class="navbar-form navbar-right" role="form">
32.  ...
33.      </form>
34.  </div>
35.  </div>
36.  </div>
37. </div>

```

- les options de menu sont obtenues par les lignes 8-29. Ce sont là encore des éléments d'une liste . La classe [active] fait que le texte est brillant indiquant par là qu'on peut cliquer sur l'option.

3.6.6 Exemple 6

Nous présenterons les médecins et les clients dans des listes déroulantes comme ci-dessous :



La liste déroulante utilisée n'est pas un composant natif Bootstrap. C'est le composant [bootstrap-select] (<http://silviomoreto.github.io/bootstrap-select/>). Pour obtenir ce résultat, nous dupliquons [app-01.html] dans [app-07.html] et nous ajoutons les ligne suivantes :

```

1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  ...
5.  <link href="bower_components/bootstrap-select/bootstrap-select.min.css" rel="stylesheet"/>
6.
7.  </head>
8.  <body>
9.  <div class="container">
10.   <h1>Rdvmedecins - v1</h1>
11.
12.   <h2><label for="medecins">Médecins</label></h2>
13.   <select id="medecins" data-style="btn btn-primary" class="selectpicker">
14.       <option value="1">Mme Marie PELISSIER</option>
15.       <option value="1">Mr Jacques BROMARD</option>
16.       <option value="1">Mr Philippe JANDOT</option>
17.       <option value="1">Mme Justine JACQUEMOT</option>
18.   </select>
19. </div>
20. <!-- Bootstrap core JavaScript ===== -->
21. ...
22. <script type="text/javascript" src="bower_components/bootstrap-select/bootstrap-select.min.js"></script>
23. <!-- script local -->

```

```

24. <script>
25.   $('.selectpicker').selectpicker();
26. </script>
27. </body>
28. </html>

```

- ligne 5 : il faut importer la feuille de style de [bootstrap-select] ;
- ligne 13 : l'attribut [data-style] est exploité par [bootstrap-select]. Il sert à donner un style à la liste déroulante. Ici, on lui donne la forme d'un bouton bleu [btn-primary] ;
- ligne 13 : l'attribut [class] est exploité ligne 23. Peut être quelconque ;
- lignes 14-17 : les éléments de la liste déroulante. On a là les balises HTML classiques ;
- ligne 22 : il faut importer le JS de [bootstrap-select] ;
- lignes 24-26 : un script JS exécuté à la fin du chargement de la page ;
- ligne 25 : une instruction JQuery. On applique la méthode [selectpicker] (**selectpicker()**) à tous les éléments ayant la classe [selectpicker] (**\$('.selectpicker')**). Il n'y en a qu'un, la balise <select> de la ligne 13. La méthode [selectpicker] vient du fichier JS référencé ligne 22 ;

3.6.7 Exemple 7

Pour afficher l'agenda d'un médecin, nous allons utiliser un tableau 'responsive' fourni par la bibliothèque JS [footable] :

Créneau horaire	Client	Action
9h00-9h20	Mme Paule MARTIN	Réserver
9h20-9h40		Supprimer

- en [1] : le tableau avec un affichage normal ;
- en [2] : le tableau lorsqu'on réduit la taille de la fenêtre du navigateur. La colonne [Action] passe automatiquement à la ligne suivante. C'est ce qu'on appelle un composant 'responsive' ou simplement adaptable.

Nous dupliquons [app-01.html] dans [app-08.html] et nous ajoutons les ligne suivantes :

```

1. ...
2. <link href="bower_components/footable/css/footable.core.min.css" rel="stylesheet"/>
3. <link href="assets/css/rdvmedecins.css" rel="stylesheet"/>
4. ...
5. <div class="container">
6.   <h1>Rdvmedecins - v1</h1>
7.
8.   <div class="row alert alert-warning">
9.     <div class="col-md-6">
10.      <table id="creneaux" class="table">
11.        <thead>
12.          <tr>
13.            <th data-toggle="true">
14.              <span>Créneau horaire</span>
15.            </th>
16.            <th>
17.              <span>Client</span>
18.            </th>
19.            <th data-hide="phone">
20.              <span>Action</span>
21.            </th>

```

```

22.      </thead>
23.      <tbody>
24.      <tr>
25.          <td>
26.              <span class='status-metro status-active'>
27.                  9h00-9h20
28.              </span>
29.          </td>
30.          <td>
31.              <span></span>
32.          </td>
33.          <td>
34.              <a href="" class="status-metro status-active">
35.                  Réserver
36.              </a>
37.          </td>
38.      </tr>
39.      <tr>
40.          <td>
41.              <span class='status-metro status-suspended'>
42.                  9h20-9h40
43.              </span>
44.          </td>
45.          <td>
46.              <span>Mme Paule MARTIN</span>
47.          </td>
48.          <td>
49.              <a href="" class="status-metro status-suspended">
50.                  Supprimer
51.              </a>
52.          </td>
53.      </tr>
54.  </tbody>
55. </table>
56. </div>
57. </div>
58. </div>
59. ...
60. <script src="bower_components/footable/dist/footable.min.js" type="text/javascript"></script>
```

- les lignes 2 et 60 sont déjà présentes dans [app-01.html]. Ce sont les fichiers CSS et JS fournis par la bibliothèque [footable] ;
- la ligne 3 référence le fichier CSS suivant :

```

@CHARSET "UTF-8";

#creneaux th {
    text-align: center;
}

#creneaux td {
    text-align: center;
    font-weight: bold;
}

.status-metro {
    display: inline-block;
    padding: 2px 5px;
    color:#fff;
}

.status-metro.status-active {
    background: #43c83c;
}

.status-metro.status-suspended {
    background: #fa3031;
}
```

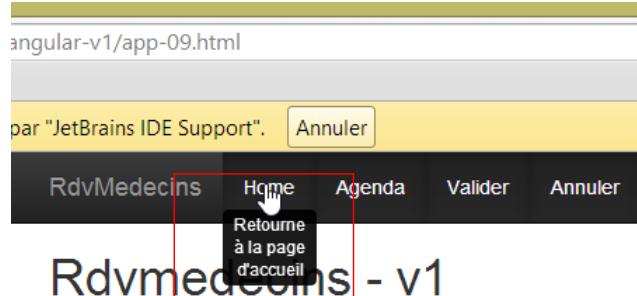
Les styles [status-*] proviennent d'un exemple d'utilisation de la table [footable] trouvé sur le site de la bibliothèque.

- ligne 8 : installe la table dans une ligne [row] et un encadré coloré [alert alert-warning] ;
- ligne 9 : la table va occuper 6 colonnes [col-md-6] ;

- ligne 10 : la table HTML est formatée par Bootstrap [class='table'] ;
- ligne 13 : l'attribut [data-toggle] indique la colonne qui héberge le symbole [+/-] qui déplie / replie la ligne ;
- ligne 19 : l'attribut [data-hide='phone'] indique que la colonne doit être cachée si l'écran a la taille d'un écran de téléphone. On peut également utiliser la valeur 'tablet' ;

3.6.8 Exemple 8

Pour aider l'utilisateur, nous allons créer des bulles d'aide (tooltip) autour des principaux composants des vues :



Pour obtenir ce résultat, nous dupliquons [app-01.html] dans [app-09.html] et nous ajoutons les lignes suivantes :

```

1. <!DOCTYPE html>
2. <html ng-app="rdvmedecins">
3. <head>
4. ...
5. </head>
6. <body>
7. <div class="container">
8.   <h1>Rdvmedecins - v1</h1>
9.   <div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
10.    <div class="container">
11.      <div class="navbar-header">
12.        <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
13.          <span class="sr-only">Toggle navigation</span>
14.          <span class="icon-bar"></span>
15.          <span class="icon-bar"></span>
16.          <span class="icon-bar"></span>
17.        </button>
18.        <a class="navbar-brand" href="#">RdvMedecins</a>
19.      </div>
20.      <!-- options de menu -->
21.      <div class="collapse navbar-collapse">
22.        <ul class="nav navbar-nav">
23.          <li class="active">
24.            <a href="">
25.              <span tooltip="Retourne à la page d'accueil" tooltip-placement="bottom">Home</span>
26.            </a>
27.          </li>
28.          <li class="active">
29.            <a href="">
30.              <span tooltip="Affiche l'agenda" tooltip-placement="top">Agenda</span>
31.            </a>
32.          </li>
33.          <li class="active">
34.            <a href="">
35.              <span tooltip="Valide le rendez-vous" tooltip-placement="right">Valider</span>
36.            </a>
37.          </li>
38.          <li class="active">
39.            <a href="">
40.              <span tooltip="Annule l'opération en cours" tooltip-placement="left">Annuler</span>
41.            </a>
42.          </li>
43.        </ul>
44.      </div>
45.    </div>
```

```

46.  </div>
47. </div>
48. <!-- Bootstrap core JavaScript ===== -->
49. <...
50. <script type="text/javascript" src="bower_components/angular-ui-bootstrap-bower/ui-bootstrap-
tpls.min.js"></script>
51. <!-- script local -->
52. <script>
53. // -----
54. angular.module("rdvmedecins", ['ui.bootstrap']);
55. </script>
56. </body>
57. </html>

```

Les bulles d'aide sont fournis par la bibliothèque [angular-ui-bootstrap] qui s'appuie elle-même sur la bibliothèque [angular]. La ligne 50 importe la bibliothèque [angular-ui-bootstrap]. Pour mettre en oeuvre les composants de la bibliothèque [angular-ui-bootstrap], il nous faut créer un module Angular. Ceci est fait aux lignes 52-55. Ces lignes définissent un module Angular nommé [rdvmedecins] (1er paramètre). Un module Angular peut utiliser d'autres modules Angular. C'est ce qu'on appelle des dépendances du module. Elles sont fournies dans un tableau comme second paramètre de la fonction [angular.module]. Ici, le module nommé [ui.bootstrap] est fourni par la bibliothèque [angular-ui-bootstrap]. C'est ce module qui va nous fournir les bulles d'aide.

La ligne 54 définit un module Angular. Par défaut, cela n'a aucun effet sur la page. On indique que la page doit être gérée par Angular, en la rattachant à un module Angular. C'est ce qui est fait, ligne 2. L'attribut [ng-app='rdvmedecins'] rattache la page au module créé ligne 54. La page va alors être analysée par Angular. Les attributs [tooltip] vont être découverts et traités par le module [ui.bootstrap].

La syntaxe de la bulle d'aide est la suivante :

```
<span tooltip="Retourne à la page d'accueil" tooltip-placement="bottom">Home</span>
```

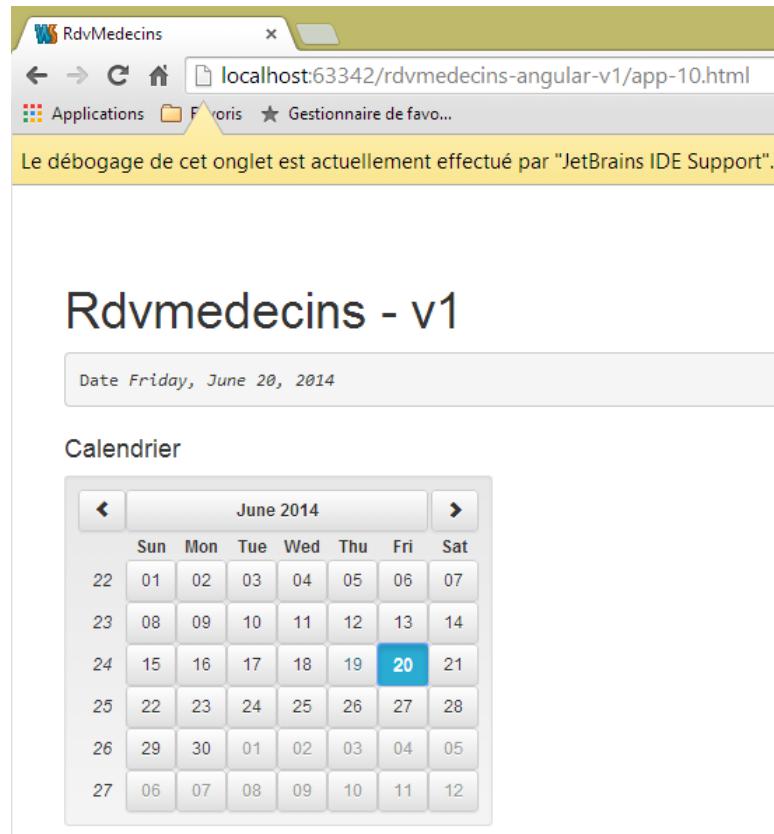
Ci-dessus, on ajoute une bulle d'aide au texte [Home] :

- [tooltip] : définit le texte de la bulle d'aide ;
- [tooltip-placement] : définit sa position (bottom, top, left, right) ;

Angular JS permet d'ajouter de nouvelles balises ou nouveaux attributs à ceux existant déjà dans le langage HTML. Cette extension du langage HTML est faite au moyen de **directives** Angular. Ici les attributs [tooltip] et [tooltip-placement] sont des attributs créés par [angular-ui-bootstrap].

3.6.9 Exemple 9

Pour aider l'utilisateur à choisir le jour d'un rendez-vous, nous allons lui proposer un calendrier :



Comme pour les bulles d'aide, ce calendrier est fourni par la bibliothèque [angular-ui-bootstrap]. Pour obtenir ce résultat, nous dupliquons [app-01.html] dans [app-10.html] et nous ajoutons les ligne suivantes :

```

1.  <!DOCTYPE html>
2.  <html ng-app="rdvmedecins">
3.  <head>
4.  ...
5.  <body>
6.  <div class="container">
7.    <h1>Rdvmedecins - v1</h1>
8.
9.    <div>
10.      <pre>Date <em>{{jour | date:'fullDate'}}</em></pre>
11.      <div class="row">
12.        <div class="col-md-2">
13.          <h4>Calendrier</h4>
14.
15.          <div style="display:inline-block; min-height:290px;">
16.            <datepicker ng-model="jour" show-weeks="true" class="well"></datepicker>
17.          </div>
18.
19.        </div>
20.      </div>
21.    </div>
22.  </div>
23. </div>
24. ...
25. <!-- script local -->
26. <script>
27.   // -----
28.   angular.module("rdvmedecins", ['ui.bootstrap'])
29. </script>
30.
31. </body>
32. </html>
```

Comme précédemment, la page est associée à un module Angular (lignes 2 et 28). Le calendrier est défini par la balise <**datepicker**> de la ligne 16 définie par la bibliothèque [angular-ui-bootstrap] :

- [show-weeks='true'] : pour afficher les n°s des semaines ;
- [class='well'] : pour entourer le calendrier d'une zone grise à coins arrondis ;
- [ng-model='jour'] : les attributs [ng-*] sont des attributs Angular. L'attribut [ng-model] désigne une donnée qui va être placée dans le modèle de la vue. Lorsque l'utilisateur va cliquer sur une date, celle-ci sera placée dans la variable [jour] du modèle. Cette variable est utilisée ligne 10. La syntaxe {{expression}} permet d'évaluer une expression composée d'éléments du modèle. Ici {{jour}} va afficher la valeur de la variable [jour] du modèle. Une caractéristique forte d'Angular est que la vue va suivre automatiquement les changements de la variable [jour]. Ainsi, lorsque l'utilisateur va changer les dates, ces changements seront immédiatement affichés ligne 10. De façon générale, le fonctionnement est le suivant :
 - une vue **V** est associée à un modèle **M** ;
 - Angular observe le modèle **M** et met **automatiquement** à jour la vue **V** lorsqu'il y a un changement de son modèle **M** ;

La syntaxe {{jour|date}} est appelée un **filtre**. Ce n'est pas la valeur de [jour] qui est affichée mais la valeur de [jour] filtrée par un filtre appelé [date]. Ce filtre est prédéfini dans Angular. Il sert à formater des dates. Il admet des paramètres précisant le format désiré. Ainsi l'expression {{jour | date:'fullDate'}} indique qu'on veut le format complet de la date, ici [Friday, June 20, 2014] parce que le calendrier est en anglais par défaut. Nous allons aborder son internationalisation prochainement.

3.6.10 Conclusion

Nous avons présenté les éléments du framework CSS Bootstrap que nous serons amenés à utiliser. C'étaient des composants passifs : leurs événements n'étaient pas gérés. Ainsi un clic sur les boutons ou les liens ne faisait rien. Ces événements seront gérés en Javascript. Il est possible d'utiliser ce langage sans l'aide de frameworks mais comme ce fut le cas côté serveur, certains frameworks s'imposent côté client. C'est le cas du framework **Angular JS** qui amène avec lui une nouvelle façon d'aborder le développement des applications Javascript exécutées par un navigateur. Nous le présentons maintenant.

3.7 Découverte d'Angular JS

Nous allons illustrer maintenant certaines des caractéristiques du framework Angular JS utilisées dans l'application. Nous en avons déjà rencontré quelques unes :

- une page HTML est propulsée par Angular JS si on lui rattache un module :

```
<html ng-app="rdvmedecins">
```

- Angular permet de créer de nouvelles balises et de nouveaux attributs HTML via des **directives** :

```
attributs : ng-app, ng-model, tooltip-placement, tooltip  
balises : datepicker
```

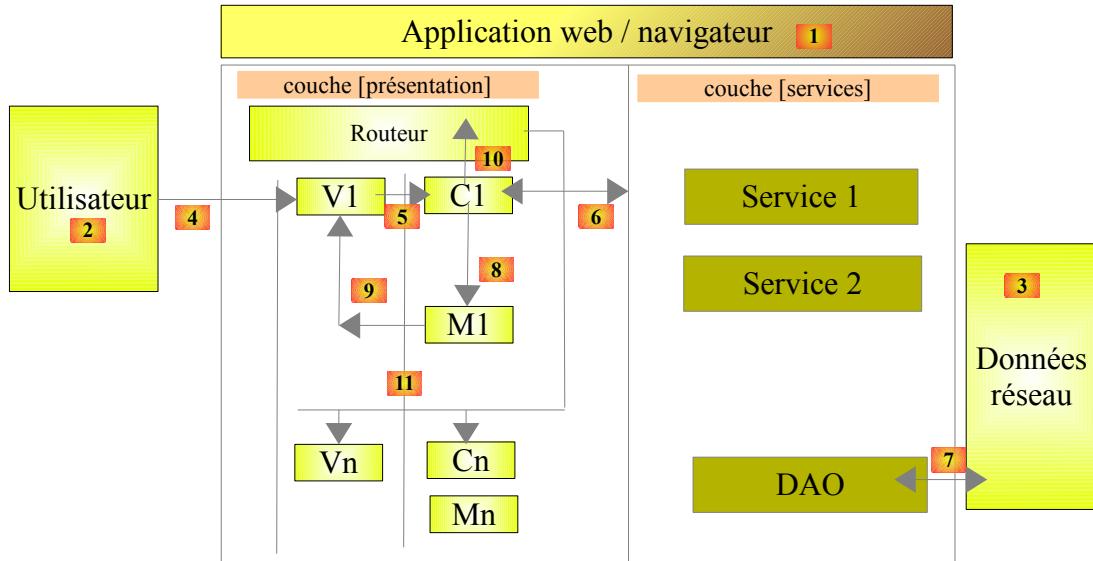
- Angular permet de créer des **filtres** :

```
{{jour|date:'fullDate'}}
```

- une vue **V** affiche un modèle **M**. Angular observe le modèle **M** et met automatiquement à jour la vue **V** lorsqu'il y a un changement de son modèle **M**. La valeur d'une variable du modèle **M** est affichée dans la vue **V** par :

```
{{variable}}
```

Nous allons commencer par approfondir l'implémentation du Design Pattern Modèle – Vue – Contrôleur dans Angular. Rappelons les liens qui existent entre-eux d'un point de vue architecture :



- la vue **V1** affiche le modèle **M1** construit par le contrôleur **C1**. Ce dernier contient non seulement le modèle **M1** mais également les gestionnaires des événements de la vue **V1**. On est dans le cycle 5, 8, 9 :
 - [5] : un événement se produit dans la vue **V1**. Il est traité par le contrôleur **C1** ;
 - celui-ci fait son travail [6-7] puis construit le modèle **M1** [8] ;
 - [9] : la vue **V1** affiche le nouveau modèle **M1**. Comme nous l'avons dit, cette dernière étape est **automatique**. Il n'y pas comme dans d'autres frameworks MVC, un **push explicite** (**C1** pousse le modèle **M1** dans **V1**) ou un **pull explicite** (la vue **V1** va chercher le modèle **M1** dans **C1**). Il y a un **push implicite** que le développeur ne voit pas ;
 - puis le cycle 5, 8, 9 reprend ;

3.7.1 Exemple 1 : le modèle MVC d'Angular

Nous allons reprendre l'exemple du calendrier. Nous avons vu la directive qui le génère :

```
<datepicker ng-model="jour" show-weeks="true" class="well"></datepicker>
```

Cette directive admet d'autres attributs que ceux présentés ci-dessus, entre-autres l'attribut [min-date] qui fixe la date minimale qu'on peut choisir dans le calendrier. Ce sera utile pour nous. Lorsque l'utilisateur choisit une date de rendez-vous, celle-ci doit être égale ou supérieure à celle du jour courant. Nous écrirons alors :

```
<datepicker ng-model="jour" ... min-date="dateMin"></datepicker>
```

où [dateMin] sera une variable du modèle de la page qui aura pour valeur la date du jour. Cela donnera la page suivante :

Rdvmedecins - v1

Date

Calendrier

June 2014						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
22	01	02	03	04	05	06
23	08	09	10	11	12	13
24	15	16	17	18	19	20
25	22	23	24	25	26	27
26	29	30	01	02	03	04
27	06	07	08	09	10	11

Rdvmedecins - v1

Date

Calendrier

June 2014						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
22	01	02	03	04	05	06
23	08	09	10	11	12	13
24	15	16	17	18	19	20
25	22	23	24	25	26	27
26	29	30	01	02	03	04
27	06	07	08	09	10	11

- en [1], nous sommes le 19 Juin 2014. Le curseur indique qu'on peut sélectionner le 19 juin ;
- en [2], le curseur indique qu'on ne peut pas sélectionner le 18 juin ;

Nous dupliquons [app-10.html] dans [app-11.html] et nous faisons les modifications suivantes :

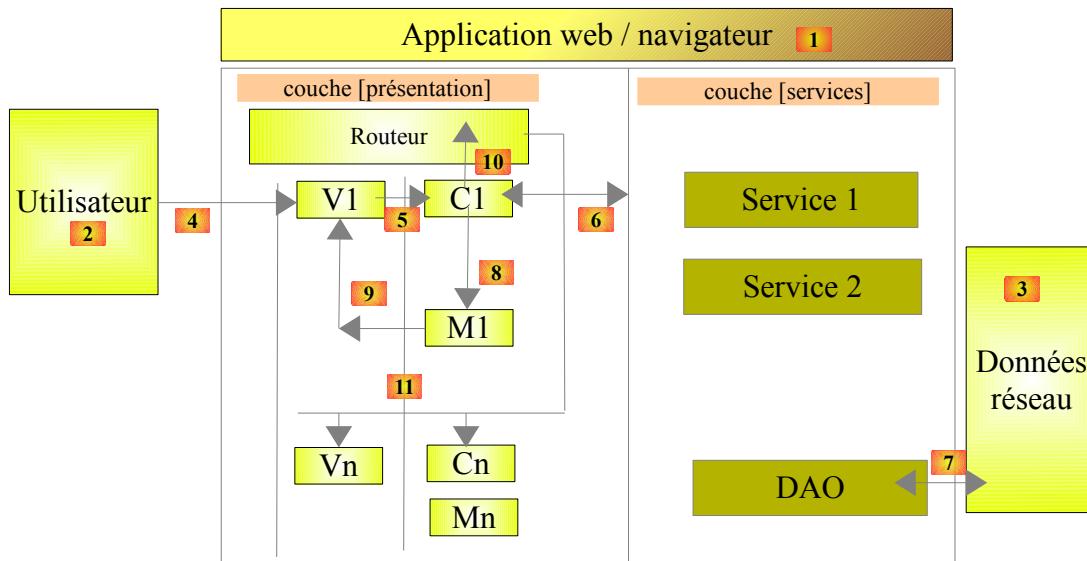
```

1. <!DOCTYPE html>
2. <html ng-app="rdvmedecins">
3. <head>
4. ...
5. </head>
6. <body ng-controller="rdvMedecinsCtrl">
7. <div class="container">
8.   <h1>Rdvmedecins - v1</h1>
9.
10.  <div>
11.    <pre>Date <em>{{jour | date:'fullDate' }}</em></pre>
12.    <div class="row">
13.      <div class="col-md-2">
14.        <h4>Calendrier</h4>
15.
16.        <div style="display:inline-block; min-height:290px;">
17.          <datepicker ng-model="jour" show-weeks="true" class="well" min-date="minDate"></datepicker>
18.        </div>
19.      </div>
20.    </div>
21.  </div>
22. </div>
23. <!-- Bootstrap core JavaScript ===== -->
24. ...
25. <!-- script local -->
26. <script>
27. // -----
28. angular.module("rdvmedecins", ['ui.bootstrap']);
29. // contrôleur
30. angular.module("rdvmedecins")
31. .controller('rdvMedecinsCtrl', ['$scope',
32.   function ($scope) {
33.     // date minimale
34.     $scope.minDate = new Date();
35.   }]);
36.
37. </script>
38.
39. </body>
40. </html>
```

Examinons d'abord le script local des lignes 26-37 :

- ligne 28 : création du module [rdvmedecins] avec sa dépendance sur le module [ui.bootstrap] qui fournit le calendrier ;
- lignes 30-35 : création d'un contrôleur. C'est lui qui va détenir le modèle de notre page. Il n'y aura pas de gestionnaire d'événement ici ;
- lignes 30-31 : le contrôleur [rdvMedecinsCtrl] appartient au module [rdvmedecins]. On peut ajouter autant de contrôleurs que l'on veut à un module. Dans notre application on aura :
 - un module de gestion de l'application ;
 - un contrôleur par vue ;
- le second paramètre de la fonction [controller] est un tableau de la forme ['O1', 'O2', ..., 'On', **function(O1, O2, ..., On)**]. Le dernier paramètre est la **fonction** qui implémente le contrôleur. Ses paramètres sont des **objets** que Angular JS va fournir à la fonction.

Revenons à l'architecture d'une application Angular :



Ci-dessus, le contrôleur C1 contient l'ensemble des gestionnaires d'événement de la vue V1 ainsi que le modèle M1 de cette dernière. Les gestionnaires d'événement peuvent avoir besoin d'un ou plusieurs services [6] pour faire leur travail. On passe l'ensemble de ceux-ci comme paramètres de la fonction de construction du contrôleur :

```
['S1', 'S2', ..., 'Sn', function(S1, S2, ..., Sn)]
```

Les services **Si** sont des singletons. Angular les crée en un unique exemplaire. Ils sont identifiés par un nom **Si**. Pourquoi sont-ils présents deux fois dans le tableau ci-dessus ? En exploitation, les scripts JS sont minifiés. Dans ce processus de minification, le tableau ci-dessus devient :

```
['S1', 'S2', ..., 'Sn', function(a1, a2, ..., an)]
```

Les paramètres perdent leur nom. Or c'est le nom de services. Il est donc important de garder ces noms. C'est pourquoi ils sont passés en tant que chaînes de caractères comme paramètres précédant la fonction. Les chaînes de caractères ne sont pas changées dans le processus de minification. Lorsqu'Angular va construire le contrôleur avec le nouveau tableau, il va remplacer **a1** par **S1**, **a2** par **S2**, ... L'ordre des paramètres est donc **important**. Il doit correspondre à l'ordre des services qui précèdent la définition de la fonction.

Revenons à la définition du contrôleur [rdvMedecinsCtrl] :

```
1. // contrôleur
2. angular.module("rdvmedecins")
3.   .controller('rdvMedecinsCtrl', ['$scope',
4.     function ($scope) {
5.       // date minimale
6.       $scope.minDate = new Date();
7.   }]);
```

- lignes 3-4 : l'unique objet injecté dans le contrôleur est l'objet **\$scope**. C'est un objet prédéfini qui représente le modèle **M** des vues associées au contrôleur. Pour enrichir le modèle d'une vue, il suffit d'ajouter des champs à l'objet **\$scope** ;
- c'est ce qui est fait ligne 6. On crée le champ [minDate] avec pour valeur la date du jour ;

La vue V exploite ce modèle M de la façon suivante :

```

1. <body ng-controller="rdvMedecinsCtrl">
2. <div class="container">
3. ...
4.     <div style="display:inline-block; min-height:290px;">
5.         <datepicker ng-model="jour" show-weeks="true" class="well" min-date="minDate"></datepicker>
6.     </div>
7. ...
8. </div>
9. ...

```

- ligne 1 : le corps de la page est associé au contrôleur [rdvMedecinsCtrl] grâce à l'attribut [ng-controller]. Cela signifie que tout ce qui est dans la balise <body> va utiliser le contrôleur [*rdvMedecinsCtrl*] pour gérer ses événements et obtenir son modèle M. Une page HTML peut dépendre de plusieurs contrôleurs imbriqués ou pas les uns dans les autres :

```

1. <div id='div1' ng-controller='c1'>
2. ...
3.     <div id='div11' ng-controller='c11'>
4. ...
5.     </div>
6. ...
7.     <div id='div12' ng-controller='c12'>
8. ...
9.     </div>
10. </div>

```

Ci-dessus :

- le contenu de [div1] (lignes 1-10) affiche le modèle **M1** géré par le contrôleur **c1**. Les balises de cette zone peuvent référencez des gestionnaires d'événement du contrôleur **c1** ;
- le contenu de [div11] (lignes 3-4) affiche le modèle **M11** géré par le contrôleur **c11** mais également le modèle **M1**. Il y a héritage des modèles. Les balises de cette zone peuvent référencez aussi bien des gestionnaires d'événement du contrôleur **c11** que des gestionnaires d'événement du contrôleur **c1**. Elles ne peuvent référencez ni le modèle **M12** du contrôleur **c12** ni les gestionnaires d'événement de celui-ci. Le contrôleur **c12** n'est en effet pas connu entre les lignes 3-5 ;
- lignes 7-9 : on peut tenir un raisonnement analogue à celui tenu précédemment ;

Revenons au code du calendrier :

```
<datepicker ng-model="jour" show-weeks="true" class="well" min-date="minDate"></datepicker>
```

L'attribut [min-date] est initialisé avec la valeur [minDate] du modèle. Implicitement `[$scope.minDate]`. Le champ est toujours cherché dans l'objet \$scope.

3.7.2 Exemple 2 : localisation des dates

Pour l'instant le calendrier ne nous est guère utile puisque c'est un calendrier anglais. Il est possible de le localiser :

Rdvmedecins - v1

Date jeudi 19 juin 2014

Calendrier [1]

June 2014

dim.	lun.	mar.	mer.	jeu.	ven.	sam.	
22	01	02	03	04	05	06	07
23	08	09	10	11	12	13	14
24	15	16	17	18	19	20	21
25	22	23	24	25	26	27	28
26	29	30	01	02	03	04	05
27	06	07	08	09	10	11	12

Langues▼

Français [2]

English

Rdvmedecins - v1

Date Thursday, June 19, 2014

Calendrier [3]

June 2014

Sun	Mon	Tue	Wed	Thu	Fri	Sat	
22	01	02	03	04	05	06	07
23	08	09	10	11	12	13	14
24	15	16	17	18	19	20	21
25	22	23	24	25	26	27	28
26	29	30	01	02	03	04	05
27	06	07	08	09	10	11	12

Langues▼

- en [1], nous avons un calendrier en français ;
- en [2], on le passe en anglais ;
- en [3], le calendrier anglais ;

Nous dupliquons la page [app-11.html] dans [app-12.html] puis nous modifions cette dernière de la façon suivante :

```

1. <!DOCTYPE html>
2. <html ng-app="rdvmedecins">
3. <head>
4. ...
5. </head>
6. <body ng-controller="rdvMedecinsCtrl">
7. <div class="container">
8.   <h1>Rdvmedecins - v1</h1>
9.
10.  <pre>Date <em>{{jour | date:'fullDate' }}</em></pre>
11.  <div class="row">
12.    <!-- le calendrier-->
13.    <div class="col-md-4">
14.      <h4>Calendrier</h4>
15.
16.      <div style="display:inline-block; min-height:290px;">
17.        <datepicker ng-model="jour" show-weeks="true" class="well" min-date="minDate"></datepicker>
18.      </div>
19.    </div>
20.    <!-- les langues -->
21.    <div class="col-md-2">
22.      <div class="btn-group" dropdown is-open="isopen">
23.        <button type="button" class="btn btn-primary dropdown-toggle" style="margin-top: 30px">
24.          Langues<span class="caret"></span>
25.        </button>
26.        <ul class="dropdown-menu" role="menu">
27.          <li><a href="" ng-click="setLang('fr')">Français</a></li>
28.          <li><a href="" ng-click="setLang('en')">English</a></li>
29.        </ul>
30.      </div>
31.    </div>
32.  </div>
33. </div>
34. ...
35. <script type="text/javascript" src="rdvmedecins.js"></script>
36. </body>
37. </html>
```

Il y a peu de modifications. Il y a simplement l'ajout lignes 21-31 de la liste déroulante des langues. Pour la première fois, nous rencontrons un gestionnaire d'événement aux lignes 27-28 :

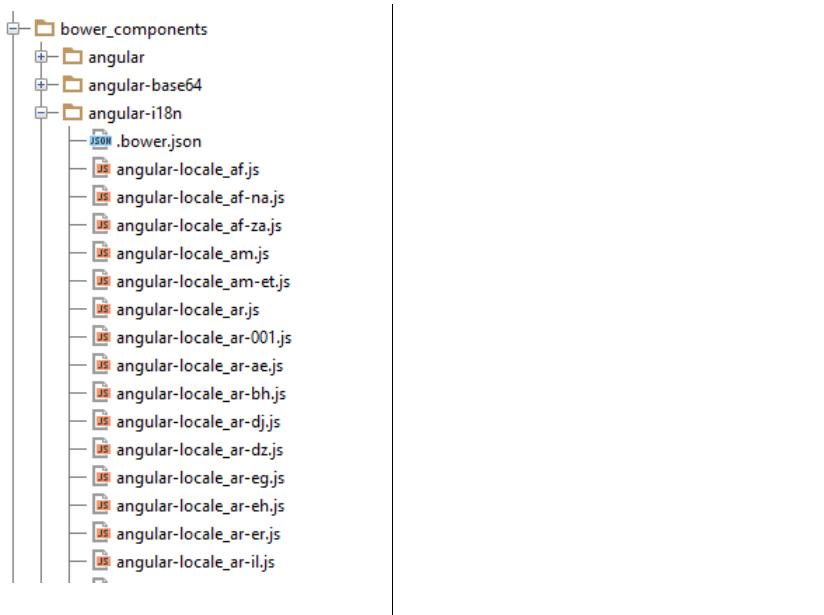
- ligne 27 : l'attribut [ng-click] est un attribut Angular qui indique le gestionnaire d'événement à exécuter lorsqu'on clique sur l'élément ayant cet attribut. Ici, la fonction [\$scope.setLang('fr')] sera exécutée. Elle mettra le calendrier en français ;
- ligne 28 : ici, on met le calendrier en anglais ;
- ligne 35 : le Javascript du contrôleur étant assez conséquent, nous le plaçons dans un fichier [rdvmedecins.js] ;

Angular gère la localisation des vues avec un module appelé [ngLocale]. La définition de notre module [rdvmedecins] sera donc la suivante :

```
1. // ----- module Angular
2. angular.module("rdvmedecins", ['ui.bootstrap', 'ngLocale']);
```

Ligne 2, il ne faut pas oublier les dépendances car Angular est parfois peu précis dans ses messages d'erreur. L'oubli d'une dépendance est ainsi particulièrement difficile à détecter. Ici on a une nouvelle dépendance sur le module [ngLocale].

Par défaut, Angular ne gère que la localisation des dates, nombres, ... qui ont des variantes locales. Il ne gère pas l'internationalisation de textes. On utilisera pour cela la bibliothèque [angular-translate]. La gestion de la localisation est faite par la bibliothèque [angular-i18n]. Cette bibliothèque amène avec elle autant de fichiers qu'il y a de variantes pour les dates, nombres, ...



Pour le calendrier français, nous utiliserons le fichier [angular-locale_fr-fr.js] et pour le calendrier anglais le fichier [angular-locale_en-us.js]. Regardons ce qu'il y a par exemple dans le fichier [angular-locale_fr-fr.js] :

```
1. 'use strict';
2. angular.module("ngLocale", [], ["$provide", function($provide) {
3.   var PLURAL_CATEGORY = {ZERO: "zero", ONE: "one", TWO: "two", FEW: "few", MANY: "many",
4.     OTHER: "other"};
5.   $provide.value("$locale", {
6.     "DATETIME_FORMATS": {
7.       "AMPMS": [
8.         "AM",
9.         "PM"
10.      ],
11.      "DAY": [
12.        "dimanche",
13.        "lundi",
14.        "mardi",
15.        "mercredi",
16.        "jeudi",
17.        "vendredi",
18.        "samedi"
19.      ],
20.      "MONTH": [
21.        "janvier",
22.        "février",
23.        "mars",
24.        "avril",
25.        "mai",
26.        "juin",
27.        "juillet",
28.        "août",
29.        "septembre",
30.        "octobre",
31.        "novembre",
32.        "décembre"
33.      ],
34.      "YEAR": [
35.        "2012",
36.        "2013",
37.        "2014",
38.        "2015",
39.        "2016",
40.        "2017",
41.        "2018",
42.        "2019",
43.        "2020"
44.      ]
45.    }
46.  }
47. });
48. });
49. 
```

```

20.    "janvier",
21.    "f\u00e9vrier",
22.    "mars",
23.    "avril",
24.    "mai",
25.    "juin",
26.    "juillet",
27.    "ao\u00fbt",
28.    "septembre",
29.    "octobre",
30.    "novembre",
31.    "d\u00e9cembre"
32.  ],
33.  "SHORTDAY": [
34.    "dim.",
35.    "lun.",
36.    "mar.",
37.    "mer.",
38.    "jeu.",
39.    "ven.",
40.    "sam."
41.  ],
42.  "SHORTMONTH": [
43.    "janv.",
44.    "f\u00e9vr.",
45.    "mars",
46.    "avr.",
47.    "mai",
48.    "juin",
49.    "juil.",
50.    "ao\u00fbt",
51.    "sept.",
52.    "oct.",
53.    "nov.",
54.    "d\u00e9c."
55.  ],
56.  "fullDate": "EEEE d MMMM y",
57.  "longDate": "d MMMM y",
58.  "medium": "d MMM y HH:mm:ss",
59.  "mediumDate": "d MMM y",
60.  "mediumTime": "HH:mm:ss",
61.  "short": "dd/MM/yy HH:mm",
62.  "shortDate": "dd/MM/yy",
63.  "shortTime": "HH:mm"
64. },
65. "NUMBER_FORMATS": {
66.   "CURRENCY_SYM": "\u20ac",
67.   "DECIMAL_SEP": ",",
68.   "GROUP_SEP": "\u00a0",
69.   "PATTERNS": [
70.     {
71.       "gSize": 3,
72.       "lgSize": 3,
73.       "macFrac": 0,
74.       "maxFrac": 3,
75.       "minFrac": 0,
76.       "minInt": 1,
77.       "negPre": "-",
78.       "negSuf": "",
79.       "posPre": "",
80.       "posSuf": ""
81.     },
82.     {

```

```

83.      "gSize": 3,
84.      "lgSize": 3,
85.      "macFrac": 0,
86.      "maxFrac": 2,
87.      "minFrac": 2,
88.      "minInt": 1,
89.      "negPre": "(",
90.      "negSuf": "\u00a0\u00a4"),
91.      "posPre": "",
92.      "posSuf": "\u00a0\u00a4"
93.    }
94.  ],
95. },
96. "id": "fr-fr",
97. "pluralCat": function (n) { if (n >= 0 && n <= 2 && n != 2) { return
  PLURAL_CATEGORY.ONE; } return PLURAL_CATEGORY.OTHER;}
98. });
99. ]]);

```

On y voit les éléments qui permettent de créer un calendrier français :

- lignes 10-18 : le tableau des jours de la semaine ;
- lignes 19-32 : le tableau des mois de l'année ;
- lignes 33-41 : le tableau des jours de la semaine en abrégé ;
- lignes 42-55 : le tableau des mois de l'année en abrégé ;
- lignes 56-63 : des formats de date et d'heure. On reconnaît ligne 62 le format 'jj/mm/aa' des dates françaises ;
- lignes 65-95 : des informations pour le formatage des nombres. Cela ne nous intéresse pas ici ;
- ligne 96 : l'identifiant 'fr-fr' de la locale du fichier (fr-fr : français de France, fr-ca : français du Canada, ...)

Dans le fichier [angular-locale_en-us.js], on a exactement la même chose mais cette fois ci pour l'anglais des USA (en-us).

Le code ci-dessus n'est pas très simple à lire. En lisant attentivement, on découvre que tout ce code définit la variable **[\$locale]** de la ligne 4. C'est en changeant la valeur de cette variable qu'on obtient l'internationalisation des dates, nombres, monnaie, ... Curieusement, Angular n'a pas prévu qu'on change la variable **[\$locale]** en cours d'exécution. On la définit une bonne fois pour toutes en important le fichier de la locale désirée :

```
<script type="text/javascript" src="bower_components/angular-i18n/angular-Locale_fr-fr.js"></script>
```

Cela ne sert à rien d'importer tous les fichiers des locales désirées, car chaque fichier, on l'a vu, ne fait qu'une chose : définir la variable **[\$locale]**. C'est le dernier fichier importé qui gagne et il n'y a ensuite plus moyen de changer la locale.

En naviguant sur la toile à la recherche d'une solution à ce problème, je n'en ai pas trouvé. J'en propose une ici [<https://github.com/stahe/angular-ui-bootstrap-datepicker-with-locale-updated-on-the-fly>]. L'idée est de mettre les différentes locales dont nous avons besoin dans un dictionnaire. C'est là que nous irons les chercher lorsqu'il faudra en changer. Le code Javascript de [rdvmedecins.js] a l'architecture suivante :

```

5      // ----- module Angular
6  angular.module("rdvmedecins", ['ui.bootstrap', 'ngLocale']);
7  // contrôleur
8  angular.module("rdvmedecins")
9    .controller('rdvMedecinsCtrl', ['$scope', '$locale',
10    function ($scope, $locale) {
11      // ----- initialisation modèle
12      // date minimale
13      $scope.minDate = new Date();
14      // locales
15      var locales = {...};
16      // on met par défaut le calendrier en français
17      angular.copy(locales['fr'], $locale);
18      // date d'aujourd'hui
19      $scope.jour=new Date();
20
21      // ----- gestionnaire d'évts
22      // changement de langue
23      $scope.setLang = function (lang) {
24        // on change la locale
25        angular.copy(locales[lang], $locale);
26        // on met à jour le jour affiché pour forcer le calendrier à changer de locale
27        $scope.jour = new Date($scope.jour.getTime());
28        // on ferme la liste déroulante
29        $scope.isopen = false;
30      };
31    }]);

```

Si on enlève la définition des locales qui prend 200 lignes (lignes 15-215 ci-dessus), le code est simple :

- ligne 6 : définit le module [rdvmedecins] et ses dépendances ;
- lignes 8-10 : définit le contrôleur [rdvMedecinsCtrl] de la page ;
- ligne 9 : la fonction de construction du contrôleur reçoit deux paramètres :
 - `$scope` : pour créer le modèle de la vue ;
 - `$locale` : qui est la variable qui gère la localisation du calendrier. C'est elle qu'il faut changer lorsqu'on change de langue ;
- ligne 13 : la variable [minDate] du modèle est initialisée avec la date du jour ;
- ligne 15 : définit le dictionnaire [locales]. Notez qu'on n'a pas écrit [`$scope.locales`]. La variable [locales] ne fait en effet pas partie du modèle exposé à la vue ;
- lignes 15-215 : définissent un dictionnaire {'fr':locale-fr-fr, 'en':locale-en-us}. Les valeurs [locale-fr-fr] et [locale-en-us] sont prises respectivement dans les fichiers JS [angular-locale_fr-fr.js] et [angular-locale_en-us.js]. Le plus dur, c'est de ne pas se tromper dans les très nombreuses parenthèses de ce dictionnaire...
- ligne 217 : on initialise la variable `$locale` avec `locales['fr']`, c-à-d la version française de la locale. On ne peut pas écrire simplement `[$locale=locales['fr']]` qui affecte à `$locale`, l'adresse de `locales['fr']`. Il faut faire une copie de valeur. Celle-ci peut se faire avec la fonction préédéfinie [angular.copy] ;
- ligne 219 : la variable [jour] du modèle est initialisée avec la date du jour. Cela entraîne que le calendrier sera affiché positionné sur cette date ;
- lignes 223-230 : définissent le gestionnaire d'événement qui est appelé lors du changement de langue. On notera la syntaxe :

```
$scope.nom_fonction=function(param1, param2, ...){...}
```

pour définir un gestionnaire d'événement qui s'appellerait [nom_fonction] et qui admettrait les paramètres [param1, param2, ...] ;

Rappelons le code HTML de la liste déroulante :

```

1.  <!-- les langues -->
2.  <div class="col-md-2">
3.    <div class="btn-group" dropdown is-open="isopen">
4.      <button type="button" class="btn btn-primary dropdown-toggle" style="margin-top: 30px">
5.        Langues<span class="caret"></span>
6.      </button>
7.      <ul class="dropdown-menu" role="menu">
8.        <li><a href="" ng-click="setLang('fr')">Français</a></li>

```

```

9.      <li><a href="" ng-click="setLang('en')">English</a></li>
10.     </ul>
11.   </div>
12. </div>

```

- ligne 8 : la sélection du français entraîne l'appel de [setLang('fr')] ;
- ligne 9 : la sélection de l'anglais entraîne l'appel de [setLang('en')] ;
- ligne 3 : l'attribut [is-open] est un booléen qui contrôle l'ouverture (true) ou la fermeture (false) de la liste déroulante. Il est initialisé avec la variable [isopen] du modèle de la vue ;

Revenons au code de [rdvmedecins.js] :

- ligne 225 : on change la valeur de la variable [\$locale] avec la valeur du dictionnaire [locales] qui convient ;
- ligne 227 : on a dit que lorsque le modèle M d'une vue V change, la vue V est automatiquement rafraîchie avec le nouveau modèle. Ligne 225, on a changé la valeur de la variable [\$locale] qui ne fait pas partie du modèle M affiché par la vue V. Il faut trouver un moyen de changer ce modèle M afin que le calendrier se rafraîchisse et utilise sa nouvelle locale. Ici, on change la variable [jour] du modèle du calendrier. On l'initialise avec un nouveau pointeur (new) qui pointe sur une date identique à celle qui est affichée. [\$scope.jour.getTime()] est le nombre de millisecondes écoulée entre le 1er janvier 1970 et la date affichée par le calendrier. Avec ce nombre, on reconstruit une nouvelle date. On va bien sûr retrouver la même date et le calendrier restera positionné sur la date qu'il affichait. Mais la valeur de [\$scope.jour] qui est en réalité un pointeur aura elle changé et le calendrier va se rafraîchir ;
- ligne 229 : on positionne à *false* la valeur de la variable [isopen] du modèle. Cette variable contrôle un des attributs de la liste déroulante :

```

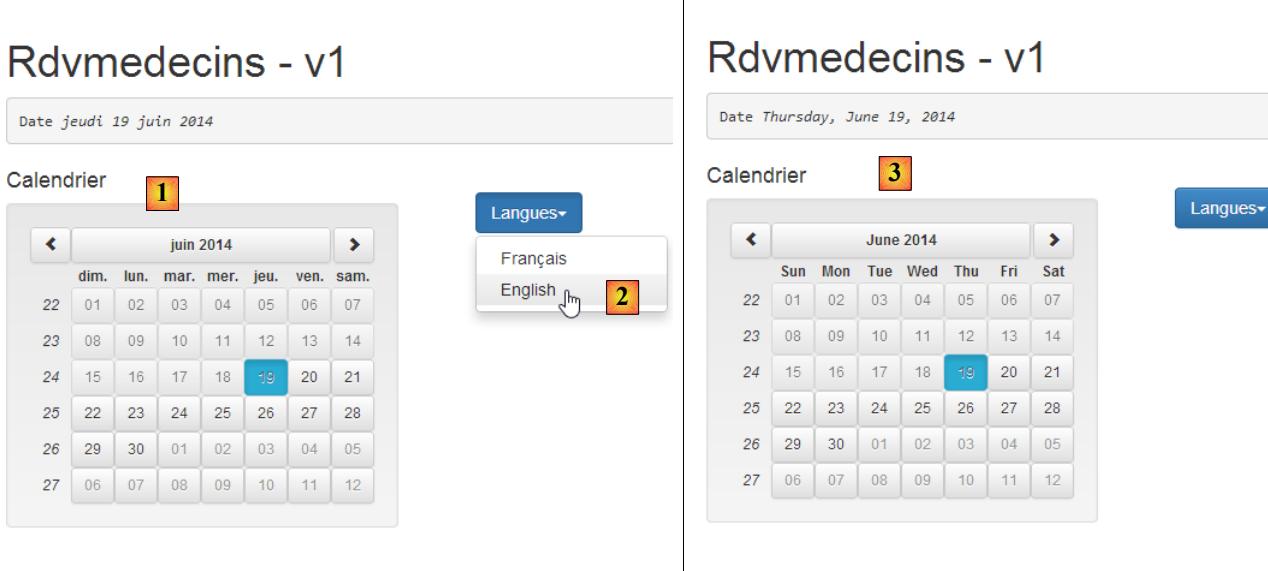
1. <div class="btn-group" dropdown is-open="isopen">
2.   <button type="button" class="btn btn-primary dropdown-toggle" style="margin-top: 30px">
3.     Langues></span>
4.   </button>
5. ...
6. </div>

```

Ligne 1 ci-dessus, l'attribut [is-open] va passer à *false*, ce qui va avoir pour effet de fermer la liste déroulante.

3.7.3 Exemple 3 : internationalisation des textes

Revenons sur la localisation du calendrier :



En [3], nous voyons que le calendrier est en anglais mais pas les textes [Calendrier, Langues]. Par défaut, Angular n'offre pas d'outil pour l'internationalisation des messages. Nous allons utiliser ici la bibliothèque [angular-translate] (<https://github.com/angular-translate/angular-translate>).

Nous allons développer l'exemple suivant :

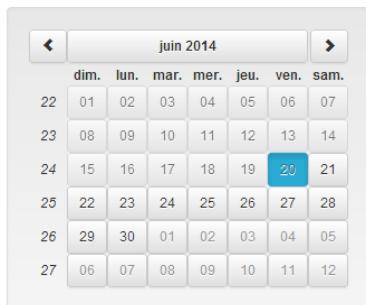
Cabinet Médical Les Médecins Associés

Agenda de Mme Laure PELISSIER
le vendredi 20 juin 2014

Aujourd'hui, il va pleuvoir...

Jour sélectionné : vendredi 20 juin 2014

Calendrier



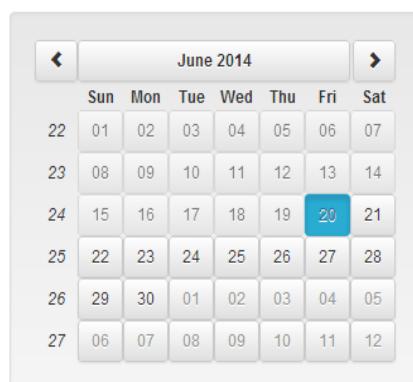
The Associated Doctors

Mme Laure PELISSIER's Diary
on Friday, June 20, 2014

Today, it will be raining...

Selected day: Friday, June 20, 2014

Calendar



- en [1], la vue en français ;
- en [2], la vue en anglais ;

Voyons la configuration nécessaire à l'internationalisation. Le script [rdvmedecins.js] est modifié de la façon suivante :

```

1. // -----
2. angular.module("rdvmedecins", ['ui.bootstrap', 'ngLocale', 'pascalprecht.translate']);
3. // configuration i18n
4. angular.module("rdvmedecins")
5.   .config(['$translateProvider', function ($translateProvider) {
6.     // messages français
7.     $translateProvider.translations("fr", {
8.       'msg_header': 'Cabinet Médical<br/>Les Médecins Associés',
9.       'msg_langues': 'Langues',
10.      'msg_agenda': 'Agenda de {{titre}} {{prenom}} {{nom}}<br/>le {{jour}}',
11.      'msg_calendrier': 'Calendrier',
12.      'msg_jour': 'Jour sélectionné : ',
13.      'msg_meteo': "Aujourd'hui, il va pleuvoir..."
14.    });
15.    // messages anglais
16.    $translateProvider.translations("en", {
17.      'msg_header': 'The Associated Doctors',
18.      'msg_langues': 'Languages',
19.      'msg_agenda': "{{titre}} {{prenom}} {{nom}}'s Diary<br/> on {{jour}}",
20.      'msg_calendrier': 'Calendar',
21.      'msg_jour': 'Selected day: ',
22.      'msg_meteo': 'Today, it will be raining...'
23.    });
24.    // langue par défaut
25.    $translateProvider.preferredLanguage("fr");
26.  }]);

```

- ligne 2 : la première modification est l'ajout d'une nouvelle dépendance. L'internationalisation de l'application nécessite le module Angular [`pascalprecht.translate`] ;
- lignes 5-26 : définissent la fonction [`config`] du module [`rdvmedecins`]. Au démarrage d'une application Angular, le framework instancie tous les services nécessaires à l'application, ceux prédéfinis d'Angular et ceux définis par l'utilisateur. Pour l'instant, nous n'avons pas défini de services. La fonction [`config`] du module d'une application est exécutée avant toute instantiation de service. Elle peut être utilisée pour définir des informations de configuration des services qui vont être ensuite instanciés. Ici, la fonction [`config`] va être utilisée pour définir les messages internationalisés de l'application ;
- ligne 5 : le paramètre de la fonction [`config`] est un tableau [`['O1', 'O2', ..., 'On', function(O1, O2, ..., On)]`] où `Oi` est un objet connu et fourni par Angular. Ici, l'objet [`$translateProvider`] est fourni par le module [`pascalprecht.translate`]. [`function`] est la fonction exécutée pour configurer l'application ;
- lignes 7-14 : la fonction [`$translateProvider.translations`] admet deux paramètres :
 - le premier paramètre est la clé d'une langue. On peut mettre ce qu'on veut. Ici, on a mis '`fr`' pour les traductions françaises (ligne 7) et '`en`' pour les traductions anglaises (ligne 16),
 - le second est la liste des traductions sous la forme d'un dictionnaire `{'cle1':'msg1', 'cle2':'msg2', ...}` ;
- lignes 7-14 : les messages français ;
- lignes 16-23 : les messages anglais ;
- ligne 25 : la méthode [`preferredLanguage`] fixe la langue par défaut. Son paramètre est l'un des arguments utilisés comme premier paramètre de la fonction [`$translateProvider.translations`] donc ici soit '`fr`' (ligne 7), soit '`en`' (ligne 16) ;
- notons qu'il y a trois sortes de messages :
 - des messages sans paramètres ni éléments HTML (lignes 9, 11, 12, ...),
 - des messages avec des éléments HTML (lignes 8, 10, ...),
 - des messages avec des paramètres (lignes 10, 19) ;

Nous dupliquons maintenant [app-11.html] dans [app-12.html] et nous faisons les modifications suivantes :

```

1. <div class="container">
2.   <!-- un premier texte avec des éléments HTML dedans -->
3.   <h3 class="alert alert-info" translate="{{'msg_header'}}"></h3>
4.   <!-- un second texte avec paramètres -->
5.   <h3 class="alert alert-warning" translate="{{msg.text}}" translate-
  values="{{msg.model}}"></h3>
6.   <!-- un troisième texte traduit par le contrôleur -->
7.   <h3 class="alert alert-danger">{{msg2}}</h3>
8.
9.   <pre>{{'msg_jour'|translate}}<em>{{jour | date:'fullDate' }}</em></pre>
10.  <div class="row">
11.    <!-- le calendrier-->
12.    <div class="col-md-4">
13.      <h4>{{'msg_calendrier'|translate}}</h4>
14.
15.      <div style="display:inline-block; min-height:290px;">
16.        <datepicker ng-model="jour" show-weeks="true" class="well" min-
  date="minDate"></datepicker>
17.      </div>
18.    </div>
19.    <!-- les langues -->
20.    <div class="col-md-2">
21.      <div class="btn-group" dropdown is-open="isopen">
22.        <button type="button" class="btn btn-primary dropdown-toggle" style="margin-top: 30px">
23.          {{'msg_langues'|translate}}<span class="caret"></span>
24.        </button>
25.        <ul class="dropdown-menu" role="menu">
26.          <li><a href="" ng-click="setLang('fr')">Français</a></li>
27.          <li><a href="" ng-click="setLang('en')">English</a></li>
28.        </ul>
29.      </div>
30.    </div>
31.  </div>
32. </div>
```

- les traductions ont lieu aux lignes 3, 5, 9, 13, 23 ;
- on peut distinguer trois syntaxes :
 - la syntaxe [`translate={{'msg_key'}}`] (ligne 3), où [`msg_key`] est une des clés d'un dictionnaire de traduction. Cette syntaxe convient aux messages avec ou sans éléments HTML mais pas à ceux avec paramètres ;

- la syntaxe `[translate="{{'msg_key'}} translate-values="{{dictionnaire}}"]` (ligne 5), convient aux messages avec ou sans éléments HTML et avec paramètres ;
- la syntaxe `["{{'msg_key'|translate}}"]` (lignes 9, 13, 23), convient aux messages sans paramètres et sans éléments HTML ;

Regardons les différents messages de cette vue :

ligne	français	anglais
3	Cabinet Médical Les Médecins Associés	The Associated Doctors
13	Calendrier	Calendar
23	Langues	Languages
9	Jour sélectionné :	Selected day:

Examinons maintenant la ligne 5 :

```
<h3 class="alert alert-warning" translate="{{msg.text}}" translate-values="{{msg.model}}"></h3>
```

On notera que `[msg.text]` et `[msg.model]` ne sont pas entourés d'apostrophes. Ce ne sont pas des chaînes de caractères mais des éléments du modèle :

- msg.text** : définit la clé du message paramétré à utiliser ;
- msg.model** : est le dictionnaire fournissant les valeurs des paramètres ;

Les noms des champs `[text, model]` peuvent être quelconques. Dans le contrôleur `[rdvMedecinsCtrl]` de la vue, l'objet `[msg]` est défini de la façon suivante :

```

rdv
30   }]);
31   // contrôleur
32   angular.module("rdvMedecins")
33   .controller('rdvMedecinsCtrl', ['$scope', '$locale', '$translate', '$filter',
34   function ($scope, $locale, $translate, $filter) {
35     // ----- initialisation modèle
36     // date minimale
37     $scope.minDate = new Date();
38     // locales
39     var locales = [...];
40     // on met par défaut le calendrier en français
41     angular.copy(locales['fr'], $locale);
42     // date d'aujourd'hui
43     $scope.jour = new Date();
44     // un texte à traduire
45     $scope.msg = {'text': 'msg_agenda', 'model': {'titre': 'Mme', 'prenom': 'Laure', 'nom': 'PELISSIER', 'jour': $filter('date')($scope.jour, 'fullDate')}};
46     // un autre texte à traduire
47     $scope.msg2 = $filter('translate')('msg_meteo');
48
49     // ----- gestionnaire d'évts
50     // changement de langue
51     $scope.setLang = function (lang) {...};
52     // observation de la variable jour
53     $scope.$watch('jour', function () {...});
54   }]);

```

- ligne 245 : la définition de l'objet `[msg]` ;
- ligne 245 : le champ `[text]` a pour valeur la clé `[msg_agenda]` qui est associée à deux valeurs :
 - `Agenda de {{titre}} {{prenom}} {{nom}}
le {{jour}}` dans le dictionnaire français ;
 - `{{titre}} {{prenom}} {{nom}}'s Diary
 on {{jour}}` dans le dictionnaire anglais ;
 Le message à afficher a donc quatre paramètres `[titre, prenom, nom, jour]` ;
- ligne 245 : le champ `[model]` est un dictionnaire donnant une valeur à ces quatre paramètres. Il y a une difficulté pour le paramètre `[jour]`. On veut afficher le nom complet du jour. Il est différent selon qu'il est en français ou en anglais. On utilise alors le filtre `[date]` déjà utilisé dans la vue sous la forme `{{ jour | date:'fullDate'}}`. Il est possible d'utiliser tout filtre dans le code Javascript sous la forme `[$filter('filter')](valeur, compléments)` où `$filter` est un objet prédéfini d'Angular et `'filter'` le nom du filtre ;
- lignes 33-34 : l'objet prédéfini `$filter` est passé comme paramètre au contrôleur, ce qui permet de l'utiliser à la ligne 245 ;

Revenons à une autre ligne de la vue affichée :

```

1.  <!-- un troisième texte traduit par le contrôleur -->
2.  <h3 class="alert alert-danger">{{msg2}}</h3>

```

Toutes les traductions précédentes se sont faites dans la vue au moyen d'attributs du module [pascalprecht.translate]. On peut décider également de faire cette traduction côté serveur. C'est ce qui est fait ici. On a dans le contrôleur (ligne 247 dans la copie d'écran ci-dessus) le code suivant :

```
$scope.msg2 = $filter('translate')('msg_meteo');
```

On utilise la même syntaxe que pour le filtre 'date' car 'translate' est lui aussi un filtre. On demande ici le message de clé 'msg_meteo'.

Examinons le mécanisme des changements de langues. On a vu que la fonction [config] de configuration du module [rdvmedecins] avait désigné le français comme langue par défaut (ligne 9 ci-dessous) :

```

1. // configuration i18n
2. angular.module("rdvmedecins")
3.   .config(['$translateProvider', function ($translateProvider) {
4.     // messages français
5.     $translateProvider.translations("fr", {...});
6.     // messages anglais
7.     $translateProvider.translations("en", {...});
8.     // langue par défaut
9.     $translateProvider.preferredLanguage("fr");
10. }]);

```

On rappelle également que la locale par défaut était également le français. Dans l'initialisation du contrôleur [rdvmedecins] on a écrit :

```

1. // on met la locale en français
2. angular.copy(locales['fr'], $locale);

```

- ligne 2 : [locales] est un dictionnaire que nous avons construit ;

Il n'y a **aucun lien** entre l'internationalisation des messages amenée par le module [pascalprecht.translate] et la localisation des dates que nous avons mise en place. Cette dernière utilise une variable \$locale qui n'est pas utilisée par le module [pascalprecht.translate]. Ce sont deux processus qui s'ignorent.

Il est maintenant temps de regarder ce qui se passe lorsque l'utilisateur change de langue :

```

248
249
250
251
252
253
254
255
256
257
258
259
260
261
262

```

```

// ----- gestionnaire d'evts
// changement de langue
$scope.setLang = function (lang) {
  // on change la locale
  angular.copy(locales[lang], $locale);
  // on met à jour le jour affiché pour forcer le calendrier à changer de locale
  $scope.jour = new Date($scope.jour.getTime());
  // on ferme la liste déroulante
  $scope.isopen = false;
  // on change la langue de traduction
  $translate.use(lang);
  // on met à jour msg2
  $scope.msg2 = $filter('translate')('msg_meteo');
};

```

- ligne 251 : lors d'un changement de langue, la fonction [setLang] est appelée avec l'un des deux paramètres ['fr','en'] ;
- lignes 252-257 : ont été déjà expliquées – elles changent la variable [\$locale] du calendrier. Cela n'a aucune incidence sur la langue des traductions ;
- ligne 259 : on change la langue des traductions. On utilise l'objet [\$translate] fourni par le module [pascalprecht.translate]. Pour cela, il faut l'injecter dans le contrôleur :

```

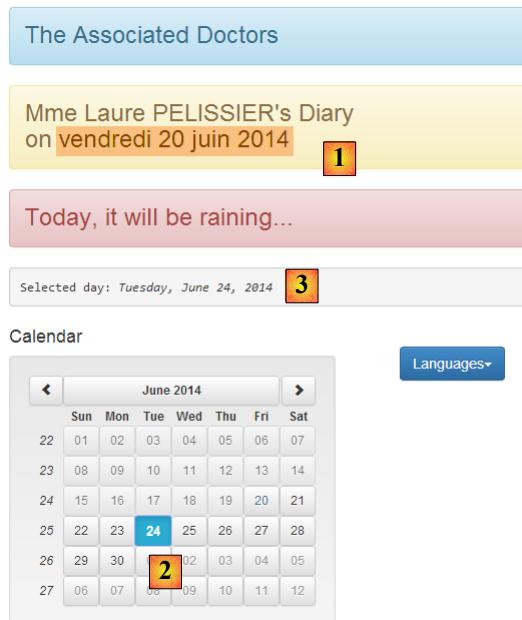
1. // contrôleur
2. angular.module("rdvmedecins")
3.   .controller('rdvMedecinsCtrl', ['$scope', '$locale', '$translate', '$filter',
4.   function ($scope, $locale, $translate, $filter) {

```

Lignes 3 et 4 ci-dessus, on injecte l'objet \$translate ;

- le paramètre `lang` de la fonction `[$translate.use(lang)]` doit avoir pour valeur l'une des clés utilisées dans la configuration comme 1er paramètre de la fonction `[$translateProvider.translations]`, c-à-d soit 'fr', soit 'en'. C'est bien le cas ;
- ligne 261 : on recalcule la valeur de `msg2`. Pourquoi ? Dans la vue, après le changement de langue opéré par la ligne 259, tous les attributs [translate] présents vont être réévalués. Ce ne sera pas le cas de l'expression `{{msg2}}` qui n'a pas cet attribut. Donc on calcule sa nouvelle valeur dans le contrôleur. Cela doit être fait après le changement de langue de la ligne 259 pour que la nouvelle langue soit utilisée pour le calcul de `[msg2]` ;

Si on s'en tient-là, on observe deux anomalies :



1. en [1], le jour est resté en français alors que le reste de la vue est en anglais ;
2. en [2] et [3], le jour sélectionné est le 24 juin alors qu'en [1], le jour reste fixé sur le 20 juin ;

Tentons des explications avant de trouver des solutions. Le message [1] est construit dans le contrôleur avec le code suivant :

```
$scope.msg = {'text': 'msg_agenda', 'model': {'titre': 'Mme', 'prenom': 'Laure', 'nom': 'PELISSIER', 'jour': $filter('date')($scope.jour, 'fullDate'))};
```

et affiché dans la vue avec le code suivant :

```
<h3 class="alert alert-warning" translate="{{msg.text}}" translate-values="{{msg.model}}"></h3>
```

L'anomalie [1] (le jour est resté en français alors que le reste de la vue est en anglais) semble montré que si l'attribut [translate] est réévalué lors d'un changement de langue, ce n'a pas été le cas de l'attribut [translate-values]. On peut alors forcer cette évaluation dans le contrôleur :

```
1.      // ----- gestionnaire d'evts
2.      // changement de langue
3.      $scope.setLang = function (lang) {
4.      ...
5.          // on met à jour msg2
6.          $scope.msg2 = $filter('translate')('msg_meteo');
7.          // et le jour de msg
8.          $scope.msg.model.jour = $filter('date')($scope.jour, 'fullDate');
9.      };
```

A chaque changement de langue, la ligne 8 ci-dessus réévalue le jour affiché. Cela règle effectivement le premier problème mais pas le second (le jour affiché dans le message ne change pas lorsqu'on sélectionne un autre jour dans le calendrier). La raison de ce comportement est la suivante. Le message est affiché dans la vue avec le code suivant :

```
<h3 class="alert alert-warning" translate="{{msg.text}}" translate-values="{{msg.model}}"></h3>
```

La vue affichée V ne change que si son modèle M change. Or ici, le choix d'un nouveau jour dans le calendrier déclenche un événement qui n'est pas géré, ce qui fait que le modèle [msg] ne change pas et que la vue donc ne change pas. Nous faisons évoluer dans la vue, la définition du calendrier :

```
<datepicker ng-model="jour" show-weeks="true" class="well" min-date="minDate"
ng-click="calendarClick()">></datepicker>
```

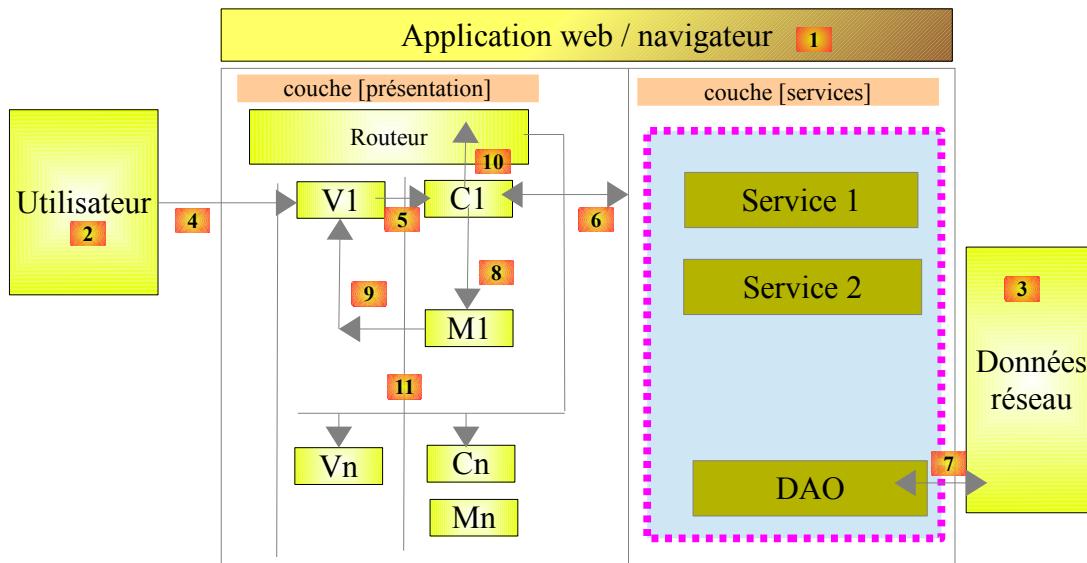
Ci-dessus, nous indiquons que le clic sur le calendrier doit être géré par la fonction `[$scope.calendarClick]`. Celle-ci est la suivante :

```
249 // ----- gestionnaire d'evts
250 // changement de langue
251 $scope.setLang = function (lang) {...};
265
266 // changement de jour dans le calendrier
267 $scope.calendarClick = function () {
268     // modification du jour affiché
269     $scope.msg.model.jour = $filter('date')($scope.jour, 'fullDate');
270 }
271 });
272
```

- ligne 267 : le gestionnaire du clic sur le calendrier ;
- ligne 269 : on force la mise à jour du jour affiché par le message [msg] ;

3.7.4 Exemple 4 : un service de configuration

Revenons sur l'architecture d'une application Angular JS :



Nous allons nous intéresser ici à la notion de service. C'est une notion assez large. Si ci-dessus, la couche [DAO] est clairement un service, tout objet Angular peut devenir un service :

- un service suit une syntaxe particulière. Il a un nom et Angular le connaît via ce nom ;
- un service peut être injecté par Angular dans les contrôleurs et les autres services ;

Certains des services que nous allons configurer dans le module [rdvmedecins] auront besoin d'être configurés. Comme un service peut être injecté dans un autre service, il est tentant de faire la configuration dans un service que nous nommerons [config] et d'injecter celui-ci dans les services et contrôleurs à configurer. Nous décrivons maintenant ce processus.

Nous dupliquons [app-13.html] dans [app-14.html] et faisons les modifications suivantes :

```
1. <div class="container">
2.   <!-- contrôle du msg d'attente -->
3.   <label>
```

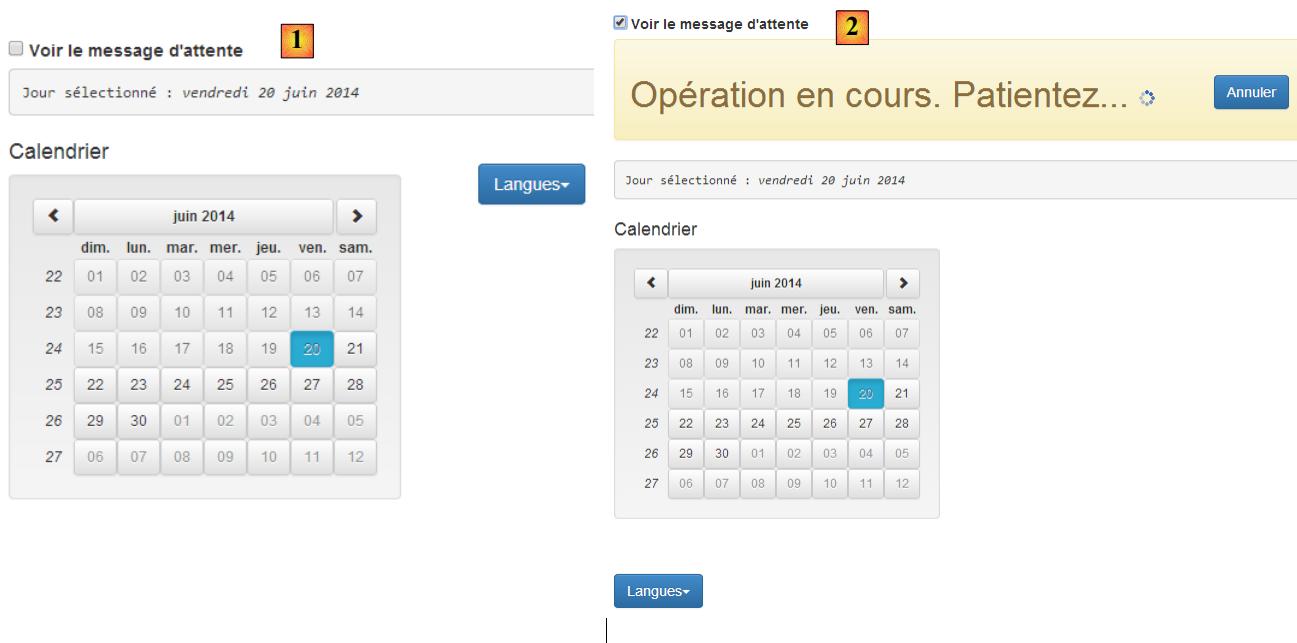
```

4.      <input type="checkbox" ng-model="waiting.visible">
5.      <span>Voir le message d'attente</span>
6.    </label>
7.
8.    <!-- le message d'attente -->
9.    <div class="alert alert-warning" ng-show="waiting.visible">
10.      <h1>{{ waiting.text | translate }}</h1>
11.      <button class="btn btn-primary pull-right" ng-click="waiting.cancel()">
12.        {{'msg_cancel'|translate}}</button>
13.      
14.    </h1>
15.  </div>
16. ...
17. </div>
18. ...
19. <script type="text/javascript" src="rdvmedecins-02.js"></script>

```

- lignes 3-6 : une case à cocher qui contrôle l'affichage ou non du message d'attente des lignes 9-15. La valeur de la case à cocher est placée dans la variable [waiting.visible] du modèle M de la vue V. Cette valeur est *true* si la case est cochée et *false* sinon. Cela marche dans les deux sens. Si nous donnons la valeur true à variable [waiting.visible], la case sera cochée. On a une association **bi-directionnelle** entre la vue V et son modèle M ;
- ligne 9-15 : un message d'attente avec un bouton d'annulation de l'attente (ligne 11) ;
- ligne 9 : le message n'est visible que si la variable [waiting.visible] a la valeur **true**. Ainsi lorsqu'on va cocher la case de la ligne 4 :
 - la valeur *true* est affectée à la variable [waiting.visible] (**ng-model**, ligne 4) ;
 - comme il y a eu changement du modèle M, la vue V est automatiquement réévaluée. Le message d'attente sera alors rendu visible (**ng-show**, ligne 9) ;
 - le raisonnement est analogue lorsqu'on décoche la case de la ligne 4 : le message d'attente est caché ;
- ligne 10 : le message d'attente est l'objet d'une traduction (filtre *translate*) ;
- ligne 11 : lorsqu'on clique sur le bouton, la méthode [waiting.cancel()] est exécutée (attribut **ng-click**) ;
- ligne 12 : le libellé du bouton fait l'objet d'une traduction ;
- ligne 19 : on met le code Javascript de l'application dans un nouveau fichier JS [rdvmedecins-02] pour ne pas perdre le code déjà écrit et qui doit être maintenant réorganisé ;

Cela donne la vue suivante :



- en [1], case non cochée ;
- en [2], case cochée ;

Le script [rdvmedecins-02] est une réorganisation du script [rdvmedecins] :

```
1  /**
2   * Created by ST on 19/06/2014.
3   */
4
5  // ----- module Angular
6 angular.module("rdvmedecins", ['ui.bootstrap', 'ngLocale', 'pascalprecht.translate']);
7
8 // configuration initiale
9 angular.module("rdvmedecins")
10 .config(['$translateProvider', function ($translateProvider) {...}]);
11
12 // service
13 angular.module("rdvmedecins")
14 .factory('config', function () {...})
15 ;
16
17 // contrôleur
18 angular.module("rdvmedecins")
19 .controller('rdvMedecinsCtrl', [...]);
20
21
```

- ligne 6 : le module [rdvmedecins] de l'application ;
- lignes 9-10 : la fonction de configuration de l'application ;
- lignes 38-39 : le service [config] ;
- lignes 283-284 : le contrôleur [rdvMedecinsCtrl] ;

Précédemment, nous avions défini dans le contrôleur le dictionnaire `locales={'fr':..., 'en': ...}` qui faisait 200 lignes. Ce dictionnaire est clairement un élément de configuration, aussi le migre-t-on dans le service [config] des lignes 38-39. Ce service est défini de la façon suivante :

```

37 // service
38 angular.module("rdvmedecins")
39   .factory('config', function () {
40     return {
41       // messages à internationaliser
42       msgWaitingInit: "msg_waiting_init",
43       msgWaiting: "msg_waiting",
44       loadingError: 'loading_error',
45       canceledOperation: 'canceled_operation',
46       getMedecinsErrors: 'get_medecins_errors',
47       getClientsErrors: 'get_clients_errors',
48       getAgendaErrors: 'get_agenda_errors',
49       selectMedecin: 'select_medecin',
50       identification: 'identification',
51       choixMedecinJourTitle: 'choixmedecinjour_title',
52       agendaTitre: 'agenda_titre',
53       selectClient: 'select_client',
54       postRemoveErrors: 'post_remove_errors',
55       resaTitre: 'resa_titre',
56       chooseAClient: 'choose_a_client',
57       postResaErrors: 'post_resa_errors',
58       // urls du client
59       urlLogin: "/login",
60       urlHome: "/home",
61       urlAgenda: "/agenda",
62       urlResa: "/resa",
63       // urls du serveur
64       urlSrvMedecins: "/getAllMedecins",
65       urlSrvClients: "/getAllClients",
66       urlSrvAgenda: "/getAgendaMedecinJour",
67       urlSrvResa: "/reserver",
68       urlSrvResaAdd: "/ajouterRv",
69       urlSrvResaRemove: "/supprimerRv",
70       // délai d'attente maximal pour les appels http en millisecondes
71       timeout: 1000,
72       // temps d'attente avant une tâche
73       waitingTimeBeforeTask: 0,
74       // mode debug
75       debug: true,
76       // le dictionnaire des locales
77       locales: {...}
78     };
79   });
80 });

```

- lignes 38-39 : un service est créé avec la fonction [factory] de l'objet [angular.module]. La syntaxe de cette fonction est comme pour les précédentes **factory('nom_service',[O1','O2', ..., 'On', function (O1, O2, ..., On){...}])** où les Oi sont les noms d'objets connus d'Angular (prédéfinis ou créés par le développeur) et qu'Angular injecte comme paramètre de la fonction **factory**. Comme ici, la fonction n'a pas de paramètres, on a utilisé une syntaxe plus courte également acceptée **factory('nom_service', function (){...})** ;
- ligne 40 : la fonction [factory] doit implémenter le service au moyen d'un objet qu'elle rend. C'est cet objet qui est le service. C'est pourquoi la fonction est-elle appelée **factory** (usine de création d'objets) ;

En général le code d'un service est de la forme :

```

1. Angular.module('nom_module')
2.   .factory('nom_service',[O1','O2', ..., 'On', function (O1, O2, ..., On){
3.     // préparation du service
4.     ...
5.     // on rend l'objet implémentant le service
6.     return {
7.       // champs
8.       ...
9.       // méthodes
10.      ...
11.    }
12.  });

```

- ligne 6 : on rend un objet JS qui peut contenir à la fois des champs et des méthodes. Ce sont ces dernières qui assurent le service ;

Ici le service [config] ne définit que des champs et aucune méthode. On y mettra tout ce qui peut être paramétré dans l'application :

- lignes 42-47 : les clés des messages à traduire ;

- lignes 59-62 : les URL de l'application ;
- lignes 64-69 : les URL du service web distant ;
- ligne 71 : un appel HTTP vers un service web qui ne répond pas, peut être long. On fixe ici à 1 seconde, le temps d'attente maximum de la réponse du service web. Passé ce délai, l'appel HTTP échoue et une exception JS est lancée ;
- ligne 73 : avant chaque appel au serveur, on va simuler une attente dont la durée est fixée ici en millisecondes. Une attente de 0 fait qu'il n'y a pas d'attente. L'application va être construite de telle façon que l'utilisateur puisse annuler une opération qu'il a lancée. Pour qu'elle puisse être annulée il faut qu'elle dure au moins quelques secondes. On utilisera cette attente artificielle pour simuler des opérations longues ;
- ligne 75 : en mode [debug=true], des informations complémentaires sont affichées dans la vue courante. Par défaut, ce mode est activé. En production, on mettrait ce champ à *false* ;
- lignes 77-278 : le dictionnaire des deux locales 'fr' et 'en'. Il était auparavant dans le contrôleur [rdvMedecinsCtrl] ;

Avec ce service, le contrôleur [rdvMedecinsCtrl] évolue de la façon suivante :

```

282 // contrôleur
283 angular.module("rdvmedecins")
284   .controller('rdvMedecinsCtrl', ['$scope', '$locale', '$translate', '$filter', 'config',
285     function ($scope, $locale, $translate, $filter, config) {
286       // ----- initialisation modèle
287       // date minimale
288       $scope.minDate = new Date();
289       // on met par défaut le calendrier en français
290       angular.copy(config.locales['fr'], $locale);
291       // date d'aujourd'hui
292       $scope.jour = new Date();
293       // message d'attente
294       $scope.waiting = {'text': config.msgWaiting, 'visible': false, 'cancel': cancel};
295
296       ...
297       $scope.setLang = function (lang) {...};
298
299       // changement de jour dans le calendrier
300       $scope.calendarClick = function () {...};
301
302       ...
303       // annulation attente
304       function cancel() {
305         // on cache le msg d'attente
306         $scope.waiting.visible = false;
307       }
308     }]);
309
310
311
312
313
314
315
316
317
318
319
320

```

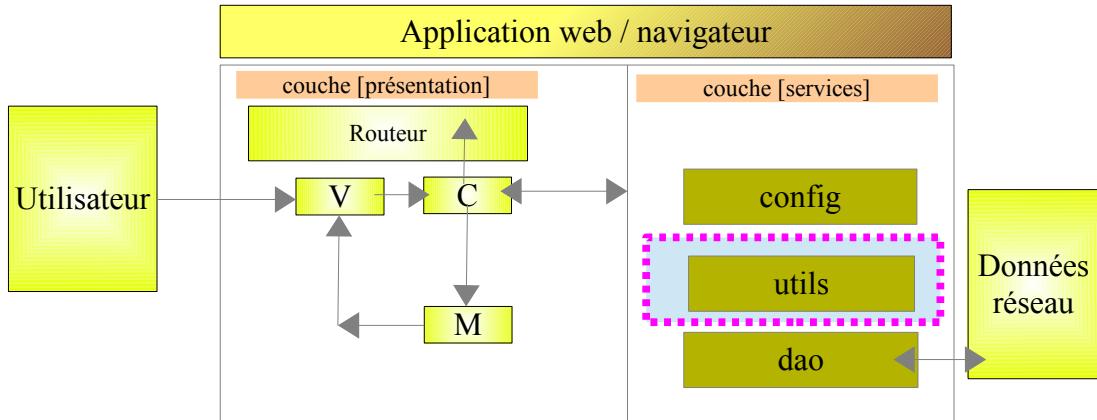
- lignes 284-285 : le service [config] est injecté dans le contrôleur ;
- ligne 290 : le dictionnaire [locales] est désormais trouvé dans le service [config] et non plus dans le contrôleur ;
- ligne 294 : l'objet [waiting] qui contrôle l'affichage du message d'attente. La clé du message d'attente est trouvée dans le service [config] (champ **text**). Par défaut le message d'attente est caché (champ **visible**). Le champ **cancel** a pour valeur, le nom de la fonction ligne 316. Ce champ est donc une méthode ou fonction ;
- ligne 316 : la fonction [cancel] est privée (on n'a pas écrit **\$scope.cancel=function(){}**). Revenons sur le code du bouton d'annulation :

```
<button class="btn btn-primary pull-right" ng-click="waiting.cancel()">
```

Lorsque l'utilisateur clique sur le bouton d'annulation, la méthode **[\$scope.waiting.cancel()]** est appelée. C'est au final la fonction privée *cancel* de la ligne 316 qui est exécutée. Elle se contente de cacher le message d'attente en mettant à false, la variable du modèle [waiting.visible] (ligne 318) ;

3.7.5 Exemple 5 : programmation asynchrone

Nous présentons maintenant un nouveau service avec une nouvelle notion, celle de la programmation asynchrone.



Notre application aura trois services :

- [**config**] : le service de configuration que nous venons de présenter ;
- [**utils**] : un service de méthodes utilitaires. Nous allons en présenter deux ;
- [**dao**] : le service d'accès au service web de prise de rendez-vous. Nous allons le présenter prochainement ;

Nous allons écrire l'application suivante :



- il s'agit de faire apparaître le bandeau [2] pendant un temps fixé par [1]. L'attente peut être annulée par [3].

Nous dupliquons [app-01.html] dans [app-15.html] et modifions le code de la façon suivante :

```

1. <!DOCTYPE html>
2. <html ng-app="rdvmedecins">
3. <head>
4.   <title>RdvMedecins</title>
5.   ...
6. </head>
7. <body ng-controller="rdvMedecinsCtrl">
8. <div class="container">
9.
10.   <!-- le message d'attente -->
11.   <div class="alert alert-warning" ng-show="waiting.visible" ng-cloak="">
12.     <h1>{{ waiting.text | translate }}</h1>
13.     <button class="btn btn-primary pull-right" ng-click="waiting.cancel()">{{ 'msg_cancel' | translate }}</button>
14.     
15.   </div>
16. </div>
17.
18.   <!-- le formulaire -->
19.   <div class="alert alert-info" ng-hide="waiting.visible">
20.     <div class="form-group">
21.       <label for="waitingTime">{{waitingTimeText | translate }}</label>

```

```
22.      <input type="text" id="waitingTime" ng-model="waiting.time"/>
23.    </div>
24.    <button class="btn btn-primary" ng-click="execute()">Exécuter</button>
25.  </div>
26. </div>
27. ..
28. <script type="text/javascript" src="rdvmedecins-03.js"></script>
29. </body>
30. </html>
```

- ligne 11 : l'attribut [ngcloak] empêche l'affichage de la zone avant que les expressions Angular de celle-ci n'aient été calculées. Cela évite un affichage bref de la zone avant l'évaluation de l'attribut [ngshow] qui va en fait provoquer sa dissimulation ;
 - ligne 22 : la saisie de l'utilisateur (temps d'attente) va être mémorisée dans le modèle [waiting.time] (attribut **ng-model**) ;
 - ligne 28 : la page utilise un nouveau script [rdvmedecins-03] ;

Le script [rdvmedecins-03] est le suivant :

```
5 // ----- module Angular
6 angular.module("rdvmedecins", ["pascalprecht.translate"]);
7
8 // configuration initiale
9 angular.module("rdvmedecins")
10 .config(['$translateProvider', function ($translateProvider) {...}]);
11
12 // services
13 angular.module("rdvmedecins")
14 .factory('config', function () {...})
15 ;
16
17 angular.module("rdvmedecins")
18 .factory('utils', ['$q', '$timeout', 'config', function ($q, $timeout, config) {...}]);
19
20 // contrôleur
21 angular.module("rdvmedecins")
22 .controller('rdvMedecinsCtrl', [...]);
```

- ligne 6 : le module Angular qui gère l'application ;
 - ligne 10 : la fonction [config] utilisée pour internationaliser les messages ;
 - ligne 41 : le service [config] que nous avons décrit ;
 - ligne 286 : le service [utils] que nous allons construire ;
 - ligne 315 : le contrôleur [rdvmedecinsCtrl] que nous allons construire ;

Nous ajoutons à la fonction [config], une nouvelle clé de message (lignes 6, 11) :

```
1. angular.module("rdvmedecins")
2.   .config(['$translateProvider', function ($translateProvider) {
3.     // messages français
4.     $translateProvider.translations("fr", {
5.     ...
6.       'msg_waiting_time_text': "Temps d'attente : "
7.     });
8.     // messages anglais
9.     $translateProvider.translations("en", {
10.    ...
11.      'msg_waiting_time_text': "Waiting time:"
12.    });
13.    // langue par défaut
14.    $translateProvider.preferredLanguage("fr");
15.  }]);
```

Nous ajoutons au service [config] une nouvelle ligne (ligne 6) pour cette clé de message :

```
1. angular.module("rdvmedecins")
2.   .factory('config', function () {
3.     return {
4.       // messages à internationaliser
5.       ...
6.       waitingTimeText: 'msg waiting time text'.
```

Le service [utils] contient deux méthodes (lignes 4, 12) :

```
1. angular.module("rdvmedecins")
2.   .factory('utils', ['$config', '$timeout', '$q', function (config, $timeout, $q) {
3.     // affichage de la représentation Json d'un objet
4.     function debug(message, data) {
5.       if (config.debug) {
6.         var text = data ? message + " : " + angular.toJson(data) : message;
7.         console.log(text);
8.       }
9.     }
10.
11.    // attente
12.    function waitForSomeTime(milliseconds) {
13.      // attente asynchrone de milliseconds milli-secondes
14.      var task = $q.defer();
15.      $timeout(function () {
16.        task.resolve();
17.      }, milliseconds);
18.      // on retourne la tâche
19.      return task;
20.    };
21.
22.    // instance du service
23.    return {
24.      debug: debug,
25.      waitForSomeTime: waitForSomeTime
26.    }
27. }]);
```

- ligne 2 : le service s'appelle [utils] (1er paramètre). Il a des dépendances sur trois services, deux services Angular prédéfinis `$timeout`, `$q` et le service `config`. Le service `[$timeout]` permet d'exécuter une fonction après qu'un certain temps se soit écoulé. Le service `[$q]` permet de créer des tâches asynchrones ;
- ligne 4 : une fonction locale `[debug]` ;
- ligne 12 : une fonction locale `[waitForSomeTime]` ;
- lignes 23-26 : l'instance du service [utils]. C'est un objet qui expose deux méthodes, celles des lignes 4 et 12. Notez que les champs de l'objet peuvent porter des noms quelconques. Par cohérence, on leur a donné les noms des fonctions qu'ils réfèrent ;
- lignes 4-9 : la méthode `[debug]` écrit sur la console un message `[message]` et éventuellement la représentation JSON d'un objet `[data]`. Cela permet d'afficher des objets de n'importe quelle complexité ;
- lignes 12-20 : la méthode `[waitForSomeTime]` crée une tâche asynchrone qui dure `[milliseconds]` milli-secondes ;
- ligne 14 : création d'une tâche grâce à l'objet prédefini `[$q]` ([https://docs.angularjs.org/api/ng/service/\\$q](https://docs.angularjs.org/api/ng/service/$q)). Ci-dessous, l'API de la tâche appelée `[deferred]` dans la documentation Angular :

The Deferred API

A new instance of `deferred` is constructed by calling `\$q.defer\(\)`.

The purpose of the `deferred` object is to expose the associated `Promise` instance as well as APIs that can be used for signaling the successful or unsuccessful completion, as well as the status of the task.

Methods

- `resolve(value)` – resolves the derived promise with the `value`. If the value is a rejection constructed via `$q.reject`, the promise will be rejected instead.
- `reject(reason)` – rejects the derived promise with the `reason`. This is equivalent to resolving it with a rejection constructed via `$q.reject`.
- `notify(value)` - provides updates on the status of the promise's execution. This may be called multiple times before the promise is either resolved or rejected.

Properties

- promise – `{Promise}` – promise object associated with this deferred.
- une tâche asynchrone `[task]` est créée par l'instruction `\[\$q.defer\(\)\]` ;
- on la termine à l'aide d'une des deux méthodes :
 - `[task.resolve(value)]` : qui termine la tâche avec `succès` et renvoie la valeur `[value]` à ceux qui attendent la fin de la tâche ;

- [task.reject(value)] : qui termine la tâche avec échec et renvoie la valeur [value] à ceux qui attendent la fin de la tâche ;

La tâche [task] peut régulièrement donner des informations à ceux qui attendent sa fin :

- [task.notify(value)] : envoie la valeur [value] à ceux qui attendent la fin de la tâche. La tâche continue à s'exécuter ;

Ceux qui veulent attendre la fin de la tâche utilisent le champ [promise] de celle-ci :

```
var promise=[task].promise ;
```

L'objet [promise] a l'API suivante (<http://www.frangular.com/2012/12/api-promise-angularjs.html>) :

Methods

- then(successCallback, errorCallback, notifyCallback) – regardless of when the promise was or will be resolved or rejected, then calls one of the success or error callbacks asynchronously as soon as the result is available. The callbacks are called with a single argument: the result or rejection reason. Additionally, the notify callback may be called zero or more times to provide a progress indication, before the promise is resolved or rejected.

This method returns a new promise which is resolved or rejected via the return value of the `successCallback`, `errorCallback`. It also notifies via the return value of the `notifyCallback` method. The promise can not be resolved or rejected from the `notifyCallback` method.

- catch(errorCallback) – shorthand for `promise.then(null, errorCallback)`
- finally(callback) – allows you to observe either the fulfillment or rejection of a promise, but to do so without modifying the final value. This is useful to release resources or do some clean-up that needs to be done whether the promise was rejected or resolved. See the [full specification](#) for more information.

Because `finally` is a reserved word in JavaScript and reserved keywords are not supported as property names by ES3, you'll need to invoke the method like `promise['finally'](callback)` to make your code IE8 and Android 2.x compatible.

Pour gérer à la fois le succès et l'échec de la tâche, on écrira :

```
1. var promise=[task].promise;
2. promise.then(successCallback, errorCallback);
3. promise['finally'](finallyCallback);
```

- ligne 1 : on récupère la promesse de la tâche ;
- ligne 2 : on définit les fonctions à exécuter en cas de succès ou en cas d'échec. On peut ne pas mettre de fonction d'échec. La fonction [successCallback] ne sera exécutée qu'à la fin de la tâche [task] avec succès [task.resolve()]. La fonction [errorCallBack] ne sera exécutée qu'à la fin de la tâche [task] avec échec [task.reject()].
- ligne 3 : on définit la fonction à exécuter après que l'une des deux fonction précédentes se soit exécutée. On met ici, le code commun aux deux fonctions [successCallback, errorCallBack].

Revenons au code de la fonction [waitForSomeTime] :

```
1. // attente
2. function waitForSomeTime(milliseconds) {
3.   // attente asynchrone de milliseconds millisecondes
4.   var task = $q.defer();
5.   $timeout(function () {
6.     task.resolve();
7.   }, milliseconds);
8.   // on retourne la tâche
9.   return task;
10.};
```

- ligne 4 : une tâche est créée ;
- lignes 5-7 : l'objet [\$timeout] permet de définir une fonction (1er paramètre) qui s'exécute après un certain délai exprimé en millisecondes (2ème paramètre). Ici le second paramètre de la fonction [\$timeout] est le paramètre de la méthode (ligne 1) ;
- ligne 6 : au bout du délai [milliseconds], la tâche est terminée avec succès ;
- ligne 9 : on retourne la tâche [task]. Il faut comprendre ici que la ligne 9 est exécutée immédiatement après la définition de l'objet [\$timeout]. On n'attend pas que le délai [milliseconds] se soit écoulé. Le code des lignes 2-10 est donc exécuté à deux moments différents :
 - une première fois qui définit l'objet [\$timeout] ;

- une seconde fois lorsque le délai [milliseconds] est écoulé ;

On a là, une fonction **asynchrone** : son résultat est obtenu à un moment ultérieur à celui de son exécution.

Le code du contrôleur qui utilise le service [config] est le suivant :

```

1. // contrôleur
2. angular.module("rdvmedecins")
3.   .controller('rdvMedecinsCtrl', ['$scope', 'utils', 'config', '$filter',
4.     function ($scope, utils, config, $filter) {
5.       // ----- initialisation modèle
6.       // message d'attente
7.       $scope.waiting = {text: config.msgWaiting, visible: false, cancel: cancel, time: undefined};
8.       $scope.waitingTimeText = config.waitingTimeText;
9.       // tâche d'attente
10.      var task;
11.      // logs
12.      utils.debug("libellé temps d'attente", $filter('translate')($scope.waitingTimeText));
13.      utils.debug("locales['fr']=", config.locales['fr']);
14.
15.      // exécution action
16.      $scope.execute = function () {
17.        // log
18.        utils.debug('début', new Date());
19.        // on affiche le msg d'attente
20.        $scope.waiting.visible = true;
21.        // attente simulée
22.        task = utils.waitForSomeTime($scope.waiting.time);
23.        // fin d'attente
24.        task.promise.then(function () {
25.          // succès
26.          utils.debug('fin', new Date());
27.        }, function () {
28.          // échec
29.          utils.debug('Opération annulée')
30.        });
31.        task.promise['finally'](function () {
32.          // fin d'attente dans tous les cas
33.          $scope.waiting.visible = false;
34.        });
35.
36.      };
37.
38.      // annulation attente
39.      function cancel() {
40.        // on termine la tâche
41.        task.reject();
42.      }
43.    }]);

```

- ligne 3 : le contrôleur utilise le service [config] ;
- ligne 7 : on a ajouté le champ [time] à l'objet [\$scope.waiting]. L'objet [\$scope.waiting.time] reçoit la valeur du délai d'attente fixé par l'utilisateur ;
- ligne 8 : la clé du message d'attente affiché par la vue est placée dans le modèle [\$scope.waitingTimeText]. De façon générale tout ce qui est affiché par une vue V doit être placé dans l'objet [\$scope] ;
- ligne 10 : une variable locale. Elle n'est pas exposée à la vue V ;
- lignes 12-13 : utilisation de la méthode [debug] du service [config]. On obtient le résultat suivant sur la console :

```

1. libellé temps d'attente : "Temps d'attente : "
2. locales['fr']= {"DATETIME_FORMATS":{"AMPMS": ["AM", "PM"], "DAY": ["dimanche", "lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi"], "MONTH": ["janvier", "février", "mars", "avril", "mai", "juin", "juillet", "août", "septembre", "octobre", "novembre", "décembre"], "SHORTDAY": ["dim.", "lun.", "mar.", "mer.", "jeu.", "ven.", "sam."], "SHORTMONTH": ["janv.", "févr.", "mars", "avr.", "mai", "juin", "juil.", "août", "sept.", "oct.", "nov.", "déc."], "fullDate": "EEEE d MMMM y", "longDate": "d MMMM y", "medium": "d MMM y HH:mm:ss", "mediumDate": "d MMM y", "mediumTime": "HH:mm:ss", "short": "dd/MM/yy HH:mm", "shortDate": "dd/MM/yy", "shortTime": "HH:mm"}, "NUMBER_FORMATS": {"CURRENCY_SYM": "€", "DECIMAL_SEP": ",", "GROUP_SEP": " ", "PATTERNS": [{"gSize": 3, "lgSize": 3, "macFrac": 0, "maxFrac": 3, "minFrac": 0, "minInt": 1, "negPre": "-", "negSuf": "", "posPre": "", "posSuf": ""}, {"gSize": 3, "lgSize": 3, "macFrac": 0, "maxFrac": 2, "minFrac": 2, "minInt": 1, "negPre": "(", "negSuf": ")", "posPre": "", "posSuf": " )"}]}, "id": "fr-fr"}

```

Ligne 2, on obtient la notation JSON de l'objet `locales['fr']`.

- ligne 16 : la méthode exécutée lorsque l'utilisateur clique sur le bouton [Exécuter] ;
- ligne 18 : affiche l'heure de début d'exécution de la méthode ;
- ligne 22 : on lance la tâche [waitForSomeTime]. On n'attend pas sa fin. L'exécution continue avec la ligne 24 suivante ;
- lignes 24-30 : on définit les fonctions à exécuter lorsque la tâche se termine avec succès (ligne 26) et en cas d'erreur (ligne 29) ;
- ligne 26 : affiche l'heure de fin d'exécution de la méthode ;
- ligne 29 : affiche que l'opération a été annulée. Cela est provoqué uniquement lorsque l'utilisateur clique sur le bouton [Annuler]. L'instruction de la ligne 41, arrête alors la tâche asynchrone avec un code d'échec ;
- lignes 31-34 : on définit la fonction à exécuter après l'exécution d'une des deux fonctions précédentes ;

Il est important de comprendre les séquences d'exécution de ce code. Dans le cas où l'utilisateur met un délai de 3 secondes et n'annule pas l'attente :

- lorsqu'il clique sur le bouton [Exécuter], la fonction `[$scope.execute]` s'exécute. Les lignes 16-34 sont exécutées sans attente des 3 secondes. A la fin de cette exécution, la vue V est synchronisée avec le modèle M. Le message d'attente est **affiché** (`ng-show=$scope.waiting.visible=true`, ligne 20) et le formulaire est **caché** (`ng-hide=$scope.waiting.visible=true`, ligne 20) ;
- à partir de ce moment l'utilisateur peut interagir de nouveau avec la vue. Il peut notamment cliquer sur le bouton [Annuler] ;
- s'il ne le fait pas, au bout de 3 secondes, la fonction du `[$timeout]` (cf lignes 5-7 ci-dessous) s'exécute :

```
1.      // attente
2.      function waitForSomeTime(milliseconds) {
3.          // attente asynchrone de milliseconds millisecondes
4.          var task = $q.defer();
5.          $timeout(function () {
6.              task.resolve();
7.          }, milliseconds);
8.          // on retourne la tâche
9.          return task;
10.     };
```

- au bout de 3 secondes donc, du code est exécuté. Ce code termine la tâche [task] avec un code de succès (`resolve`). Cela va déclencher l'exécution de tous les codes qui attendaient cette fin (ligne 4 ci-dessous) :

```
1.      // attente simulée
2.      task = utils.waitForSomeTime($scope.waiting.time);
3.      // fin d'attente
4.      task.promise.then(function () {
5.          // succès
6.          utils.debug('fin', new Date());
7.      }, function () {
8.          // échec
9.          utils.debug('Opération annulée')
10.     });
11.     task.promise['finally'](function () {
12.         // fin d'attente dans tous les cas
13.         $scope.waiting.visible = false;
14.     });
15.    );
```

- la ligne 6 ci-dessus (fin avec succès) va donc être exécutée. Puis ce sera le tour des lignes 11-14. Une fois ce code exécuté, on revient à la vue V qui va alors être synchronisée avec son modèle M. Le message d'attente est **caché** (`ng-show=$scope.waiting.visible=false`, ligne 13) et le formulaire est **affiché** (`ng-hide=$scope.waiting.visible=false`, ligne 13) ;

Les affichages écran sont alors les suivants :

```
début : "2014-06-23T15:05:58.480Z"
fin : "2014-06-23T15:06:01.481Z"
```

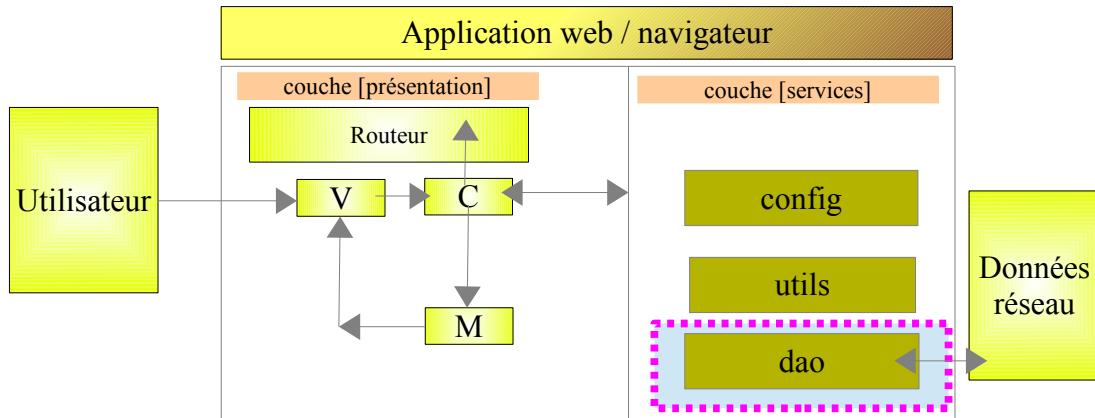
On voit ci-dessus, le délai de 3 secondes (06:01-05:58) entre le début et la fin de l'attente. Si on contraire, l'utilisateur annule l'attente avant les 3 secondes, on a l'affichage suivant :

```
début : "2014-06-23T15:08:09.564Z"
Opération annulée
```

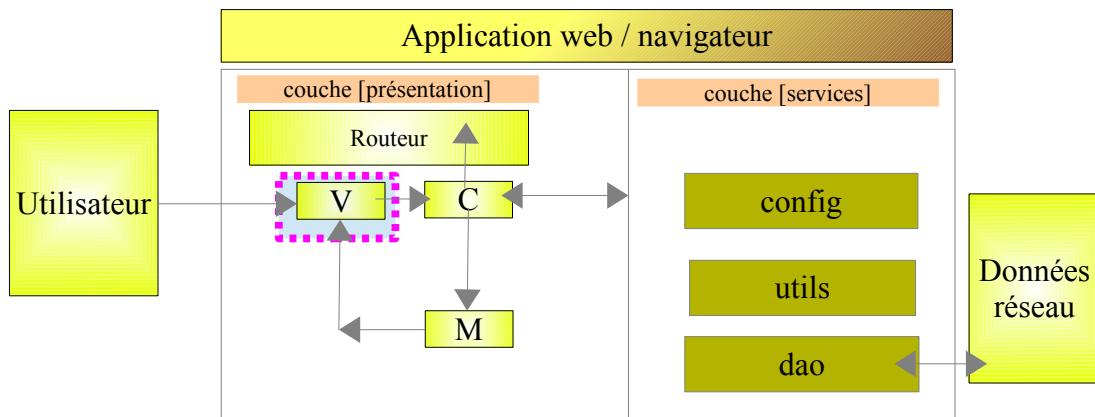
Pour terminer, il est important de comprendre **qu'à tout moment il n'y a qu'un thread d'exécution** appelé le thread de l'UI (User Interface). La fin d'une tâche asynchrone est signalée par un événement exactement comme l'est le clic sur un bouton. Cet événement n'est pas traité immédiatement. Il est mis dans la file d'attente des événements qui attendent leur exécution. Lorsque vient son tour, il est traité. Ce traitement utilise le thread de l'UI et donc pendant ce temps, l'interface est gelée. Elle ne réagit pas aux sollicitations de l'utilisateur. Pour cela, il est important que le traitement d'un événement soit rapide. Parce que chaque événement est traité par le thread de l'UI, on n'a jamais à régler des problèmes de synchronisation entre threads s'exécutant en même temps. Il n'y a, à chaque instant, que le thread de l'UI qui s'exécute.

3.7.6 Exemple 6 : les services HTTP

Nous présentons maintenant le service [dao] qui communique avec le serveur web :



3.7.6.1 La vue V



Nous allons écrire un formulaire pour demander la liste des médecins :

Rdvmedecins - v1

Temps d'attente :

URL du service web :

Login :

Mot de passe :

Liste des médecins

Les erreurs suivantes se sont produites lors du chargement de la liste des médecins

- L'accès au serveur n'a pu être réalisé. Vérifiez sa disponibilité

Nous dupliquons [app-01.html] dans [app-16.html] que nous modifions ensuite de la façon suivante :

```
1. <div class="container" ng-cloak="">
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <!-- le message d'attente -->
5.   <div class="alert alert-warning" ng-show="waiting.visible" ng-cloak="">
6.     <h1>{{ waiting.text | translate }}</h1>
7.     <button class="btn btn-primary pull-right" ng-click="waiting.cancel()">{{'msg_cancel'|translate}}</button>
8.     
9.   </h1>
10.  </div>
11.
12.  <!-- la demande -->
13.  <div class="alert alert-info" ng-hide="waiting.visible">
14.    <div class="form-group">
15.      <label for="waitingTime">{{waitingTimeText | translate}}</label>
16.      <input type="text" id="waitingTime" ng-model="waiting.time"/>
17.    </div>
18.    <div class="form-group">
19.      <label for="urlServer">{{urlServerLabel | translate}}</label>
20.      <input type="text" id="urlServer" ng-model="server.url"/>
21.    </div>
22.    <div class="form-group">
23.      <label for="Login">{{loginLabel | translate}}</label>
24.      <input type="text" id="Login" ng-model="server.Login"/>
25.    </div>
26.    <div class="form-group">
27.      <label for="password">{{passwordLabel | translate}}</label>
28.      <input type="password" id="password" ng-model="server.password"/>
29.    </div>
30.    <button class="btn btn-primary" ng-click="execute()">{{medecins.title|translate:medecins.model}}</button>
31.  </div>
32.
33.  <!-- la liste des médecins -->
34.  <div class="alert alert-success" ng-show="medecins.show">
35.    {{medecins.title|translate:medecins.model}}
36.    <ul>
37.      <li ng-repeat="medecin in medecins.data">{{medecin.titre}} {{medecin.prenom}} {{medecin.nom}}</li>
38.    </ul>
39.  </div>
40.
41.  <!-- la liste d'erreurs -->
42.  <div class="alert alert-danger" ng-show="errors.show">
43.    {{errors.title|translate:errors.model}}
44.    <ul>
45.      <li ng-repeat="message in errors.messages">{{message|translate}}</li>
46.    </ul>
```

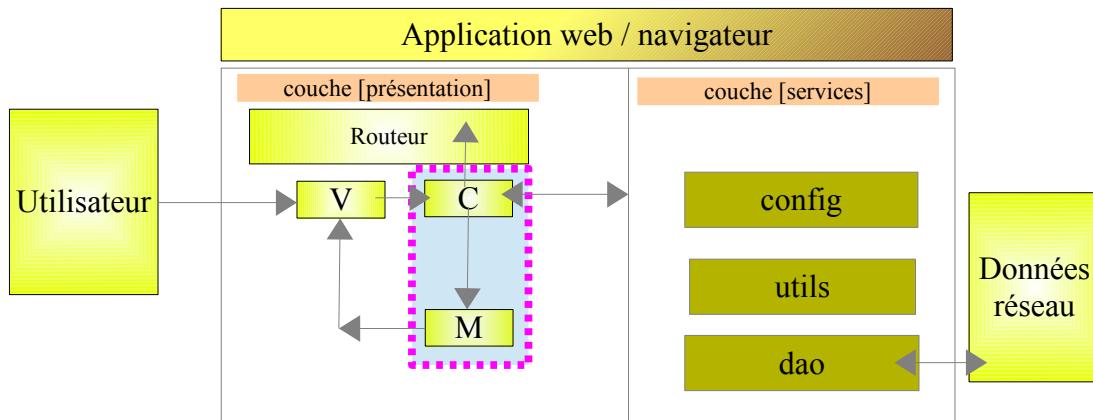
```

47.  </div>
48.
49. </div>
50. ...
51. <script type="text/javascript" src="rdvmedecins-04.js"></script>

```

- lignes 13-31 : implémentent le formulaire. Celui-ci n'est pas visible lorsque le message d'attente est affiché (`ng-hide="waiting.visible"`). On retiendra que les quatre saisies sont mémorisées dans (attributs `ng-model`) [waiting.time (ligne 16), server.url (ligne 20), server.login (ligne 24), server.password (ligne 28)] ;
- lignes 34-39 : affichent la liste des médecins. Cette liste n'est pas toujours visible (`ng-show="medecins.show"`).
- ligne 35 : une alternative à la syntaxe `<div ... translate="{{medecins.title}}" translate-values="{{medecins.model}}>` déjà rencontrée ;
- ligne 36 : une liste non ordonnée ;
- ligne 37 : la liste des médecins sera trouvée dans le modèle [medecins.data]. La directive Angular [`ng-repeat`] permet de parcourir une liste. La syntaxe `ng-repeat="medecin in medecins.data"` demande à ce que la balise `` soit répétée pour chaque élément de la liste [medecins.data]. L'élément courant de la liste est appelée [medecin] ;
- ligne 37 : pour chaque ``, on écrit le titre, le prénom et le nom du médecin courant désigné par la variable [medecin] ;
- lignes 42-47 : affichent la liste des erreurs. Cette liste n'est pas toujours visible (`ng-show="errors.show"`). Cet affichage suit le même modèle que l'affichage de la liste des médecins. De façon générale, pour afficher une liste d'objets, on utilise la directive Angular [`ng-repeat`] ;
- ligne 51 : le code Javascript est maintenant dans le fichier [rdvmedecins-04]

3.7.6.2 Le contrôleur C et le modèle M



Le code Javascript évolue de la façon suivante :

```
5 // ----- module Angular
6 angular.module("rdvmedecins", [
7   "pascalprecht.translate",
8   "base64"
9 ]);
10
11 // configuration initiale
12 angular.module("rdvmedecins")
13 .config(['$translateProvider', function ($translateProvider) {...}]);
14
15 // services
16 angular.module("rdvmedecins")
17 .factory('config', function () {...})
18 ;
19
20 angular.module("rdvmedecins")
21 .factory('utils', ['$config', '$timeout', '$q', function (config, $timeout, $q) {...}]);
22
23 angular.module("rdvmedecins")
24 .factory('dao', [...]);
25
26 // contrôleur
27 angular.module("rdvmedecins")
28 .controller('rdvMedecinsCtrl', [...])
29 ;
```

- lignes 6-9 : le module [rdvmedecins] déclare une dépendance sur le module [base64] fourni par la bibliothèque [angular-base64] qui est l'une des dépendances du projet. Ce module sert à coder en Base64, la chaîne [login:password] envoyée au service web pour s'authentifier ;
 - lignes 12-13 : la fonction d'initialisation qui contient nos messages internationalisés. De nouveaux messages apparaissent. Nous ne les présenterons plus ;
 - lignes 69-70 : le service [config] qui paramètre notre application. De nouvelles clés de message y sont ajoutées. Nous ne les présenterons plus ;
 - lignes 318-319 : le service [utils] qui contient des méthodes utilitaires. De nouvelles y sont rajoutées. Nous les présenterons ;
 - lignes 385-386 : le service [dao] chargé des échanges avec le service web. C'est sur lui que nous allons nous concentrer ;
 - lignes 467-468 : le contrôleur C de la vue V que nous venons de présenter. Nous allons le présenter maintenant car c'est lui le chef d'orchestre qui réagit aux demandes de l'utilisateur ;

3.7.6.3 Le contrôleur C

Le code du contrôleur est le suivant :

```
1. angular.module("rdvmedecins")
2.   .controller('rdvMedecinsCtrl', ['$scope', 'utils', 'config', 'dao', '$translate',
3.     function ($scope, utils, config, dao, $translate) {
4.       // ----- initialisation modèle
5.       // modèle
6.       $scope.waiting = {text: config.msgWaiting, visible: false, cancel: cancel, time: undefined};
7.       $scope.waitingTimeText = config.waitingTimeText;
8.       $scope.server = {url: undefined, login: undefined, password: undefined};
9.       $scope.medecins = {title: config.listMedecins, show: false, model: {}};
10.      $scope.errors = {show: false, model: {}};
11.      $scope.urlServerLabel = config.urlServerLabel;
12.      $scope.loginLabel = config.loginLabel;
13.      $scope.passwordLabel = config.passwordLabel;
14.
15.      // tâche asynchrone
16.      var task;
17.
18.      // exécution action
19.      $scope.execute = function () {
20.        // on met à jour l'UI
21.        $scope.waiting.visible = true;
22.        $scope.medecins.show = false;
23.        $scope.errors.show = false;
24.        // attente simulée
```

```

25.     task = utils.waitForSomeTime($scope.waiting.time);
26.     var promise = task.promise;
27.     // attente
28.     promise = promise.then(function () {
29.         // on demande la liste des médecins;
30.         task = dao.getData($scope.server.url, $scope.server.login, $scope.server.password,
config.urlSrvMedecins);
31.         return task.promise;
32.     });
33.     // on analyse le résultat de l'appel précédent
34.     promise.then(function (result) {
35.         // result={err: 0, data: [med1, med2, ...]}
36.         // result={err: n, messages: [msg1, msg2, ...]}
37.         if (result.err == 0) {
38.             // on met les données acquises dans le modèle
39.             $scope.medecins.data = result.data;
40.             // on met à jour l'UI
41.             $scope.medecins.show = true;
42.             $scope.waiting.visible = false;
43.         } else {
44.             // il y a eu des erreurs pour obtenir la liste des médecins
45.             $scope.errors = { title: config.getMedecinsErrors, messages:
utils.getErrors(result), show: true, model: {}};
46.             // on met à jour l'UI
47.             $scope.waiting.visible = false;
48.         }
49.     });
50. };
51.
52. // annulation attente
53. function cancel() {
54.     // on termine la tâche
55.     task.reject();
56.     // on met à jour l'UI
57.     $scope.waiting.visible = false;
58.     $scope.medecins.show = false;
59.     $scope.errors.show = false;
60. }
61.
62. }
63. ])
64. ;

```

- ligne 2 : le contrôleur a une nouvelle dépendance, celle sur le service [dao] ;
- lignes 6-13 : le modèle M de la vue V est initialisé pour le 1er affichage de celle-ci ;
- ligne 8 : [\$scope.server] va être utilisé pour récupérer trois des quatre informations du formulaire V, la quatrième étant mémorisée dans [\$scope.waiting.time] (ligne 6) ;
- ligne 9 : [\$scope.medecins] va rassembler les informations nécessaires à l'affichage de la liste des médecins :

```

1.   <!-- la liste des médecins -->
2.   <div class="alert alert-success" ng-show="medecins.show">
3.     {{medecins.title|translate:medecins.model}}
4.     <ul>
5.       <li ng-repeat="medecin in medecins.data">{{medecin.titre}} {{medecin.prenom}}
{{medecin.nom}}</li>
6.     </ul>
7.   </div>

```

L'attribut [medecins.title] sera le titre du bandeau. Il est défini dans le service [config]. L'attribut [medecins.show] va contrôler l'affichage ou non du bandeau (attribut **ng-show="medecins.show"**). L'attribut [medecins.model] est un dictionnaire vide et le restera. Il sert simplement à illustrer l'utilisation de la variante de traduction utilisée ligne 3. Non défini encore, l'attribut [medecins.data] qui contiendra la liste des médecins (ligne 5).

- ligne 10 : [\$scope.errors] va rassembler les informations nécessaires à l'affichage de la liste des erreurs :

```

1.   <!-- la liste d'erreurs -->
2.   <div class="alert alert-danger" ng-show="errors.show">

```

```

3.    {{errors.title|translate:errors.model}}
4.    <ul>
5.      <li ng-repeat="message in errors.messages">{{message|translate}}</li>
6.    </ul>
7.  </div>

```

L'attribut [errors.title] sera le titre du bandeau. Il est défini dans le service [config]. L'attribut [errors.show] va contrôler l'affichage ou non du bandeau (attribut **ng-show="errors .show"**). L'attribut [errors.model] est un dictionnaire vide et le restera. Il sert simplement à illustrer l'utilisation de la variante de traduction utilisée ligne 3. Non défini encore, l'attribut [errors.messages] qui contiendra la liste des messages d'erreur à afficher (ligne 5).

- ligne 16 : la tâche asynchrone. Le contrôleur va lancer successivement deux tâches asynchrones. Les références sur ces tâches successives seront placées dans la variable [task]. Cela permettra de les annuler (ligne 55) ;
- ligne 19 : la méthode exécutée lorsque l'utilisateur clique sur le bouton [Liste des médecins] :

```
<button class="btn btn-primary" ng-click="execute()">Liste des médecins</button>
```

- lignes 21-23 : l'interface visuelle est mise à jour : le message d'attente est affiché, tout le reste est caché ;
- ligne 25 : on crée la tâche asynchrone de l'attente. On recevra un signal (tâche réalisée) au bout du temps saisi par l'utilisateur dans le formulaire ;
- ligne 26 : on récupère la promesse de la tâche asynchrone. C'est avec elle que le programme qui lance la tâche travaille. Il faut cependant avoir la référence de la tâche elle-même afin de pouvoir l'annuler (ligne 55) ;
- lignes 28-32 : on définit le travail à faire lorsque l'attente sera terminée ;
- ligne 30 : on utilise la méthode [dao.getData] pour lancer une nouvelle tâche asynchrone. On lui passe les informations dont elle a besoin :
 - l'URL racine du service web [\$scope.server.url], par exemple [http://localhost:8080];
 - le login [\$scope.server.login] pour s'identifier, par exemple [admin];
 - le mot de passe [\$scope.server.password] pour s'identifier, par exemple [admin];
 - l'URL qui rend le service demandé [config.urlSvrMedecins], ici [/getAllMedecins]. Au total l'URL complète sera [http://localhost:8080/getAllMedecins] ;

La méthode [dao.getData] rend un résultat qui a deux formes possibles :

- {err: 0, data: [med1, med2, ...]} où [medi] est un objet représentant un médecin (titre, prenom, nom),
- {err: n, messages: [msg1, msg2, ...]} où [msgi] est un message d'erreur et n est différent de 0 ;

- ligne 31 : on retourne la promesse de la tâche. Là il y a quelque chose à comprendre. On a deux promesses :
 - promise.then()** : rend une première promesse [promise1] ;
 - return task.promise** : rend une seconde promesse [promise2] ;
 - au final **promise=promise.then(... ;return task.promise)** est une chaîne de deux promesses [promise2.promise1]. [promise1] ne sera évaluée que lorsque la promesse [promise2] sera obtenue, c-à-d lorsque la tâche [dao.getData] sera terminée. La promesse [promise1] ne dépend d'aucune tâche asynchrone. Elle sera donc obtenue immédiatement ;
- lignes 34-50 : de l'explication précédente, il découle que ces lignes se seront exécutée que lorsque la tâche [dao.getData] sera terminée. Le paramètre [result] passé à la fonction de la ligne 34 est construit par la méthode [dao.getData] et transmis au code appelant par l'opération [**task.resolve(result)**] où [result] est de la forme suivante :
 - {err: 0, data: [med1, med2, ...]} où [medi] est un objet représentant un médecin (titre, prenom, nom),
 - {err: n, messages: [msg1, msg2, ...]} où [msgi] est un message d'erreur et n est différent de 0 ;
- ligne 37 : on regarde le code d'erreur [result.err] ;
- lignes 38-42 : si l'il n'y a pas d'erreur (result.err==0), alors on récupère la liste des médecins et on l'affiche ;
- lignes 44-47 : si au contraire il y a erreur (result.err !=0), alors on récupère la liste des messages d'erreurs et on l'affiche ;
- lignes 53-56 : le message d'attente avec son bouton d'annulation est présent tant que les deux opérations asynchrones ne sont pas terminées. Voyons ce qui se passe selon le moment de l'annulation :
 - il faut tout d'abord comprendre que les lignes 19-50 sont exécutées d'une traite. Une seule tâche asynchrone a alors été lancée, celle de la ligne 25,
 - après cette première exécution, la vue V est mise à jour et donc le bandeau d'attente et son bouton d'annulation est visible. Si l'utilisateur annule l'attente avant que la tâche de la ligne 25 ne soit terminée, la méthode de la ligne 53 est alors exécutée et la tâche est annulée avec échec (ligne 55) ;
 - lignes 56-59 : l'interface est mise à jour : on réaffiche le formulaire et tout le reste est caché,
 - il y a alors retour à la vue V et le navigateur va traiter l'événement suivant. Puisqu'il y a eu fin de tâche, la promesse de cette tâche est obtenue, ce qui crée un événement. Il est alors traité ;
 - les lignes 28-32 sont ensuite exécutées. Il n'y a pas de fonction définie pour le cas d'échec, donc aucun code n'est exécuté. On obtient une nouvelle promesse, celle toujours rendue par [promise.then] et toujours obtenue,

- l'événement ayant été traité, il y a retour à la vue V et le navigateur va traiter l'événement suivant. Puisque la [promise] de la ligne 28 a été traitée, celle de la ligne 34 va être résolue, ce qui va provoquer un nouvel événement. Il est alors traité ;
- les lignes 34-49 vont alors être exécutées à leur tour, car la promesse utilisée ligne 34 a été obtenue. De nouveau, parce qu'il n'y a pas de fonction définie pour le cas d'échec, aucun code n'est exécuté,
- on arrive ainsi à la ligne 50. Il n'y a plus d'attente de tâche et la nouvelle vue V est affichée ;
- supposons maintenant que l'annulation intervient pendant que la seconde tâche asynchrone [dao.getData] est en cours d'exécution. Le raisonnement précédent peut être tenu de nouveau. La fin de la tâche va provoquer l'exécution des lignes 34-50 avec une fin de tâche avec échec. On va découvrir bientôt que la méthode [dao.getData] réalise un appel HTTP asynchrone vers le service web. Cet appel ne sera pas annulé mais son résultat ne sera pas exploité.

Il est important de comprendre ce va et vient constant entre l'affichage de la vue V et le traitement des événements du navigateur. Les événements sont provoqués par l'utilisateur (un clic) ou par des opérations système telles que la fin d'une opération asynchrone. L'état de repos du navigateur est l'affichage de la vue V. Il est tiré de ce repos par un événement qui se produit et qu'il traite alors. Dès que l'événement a été traité, il revient à son état de repos. La vue V est alors mise à jour si l'événement traité a modifié son modèle M. Le navigateur est tiré de son état de repos par l'événement suivant.

Tout se passe dans un unique thread. Deux événements ne sont jamais traités simultanément. Leur exécution est séquentielle. Le navigateur ne passe à l'événement suivant que lorsque le précédent lui laisse la main, en général parce qu'il a été traité totalement.

Il nous reste un point à expliquer. Pour afficher les messages d'erreur, nous écrivons :

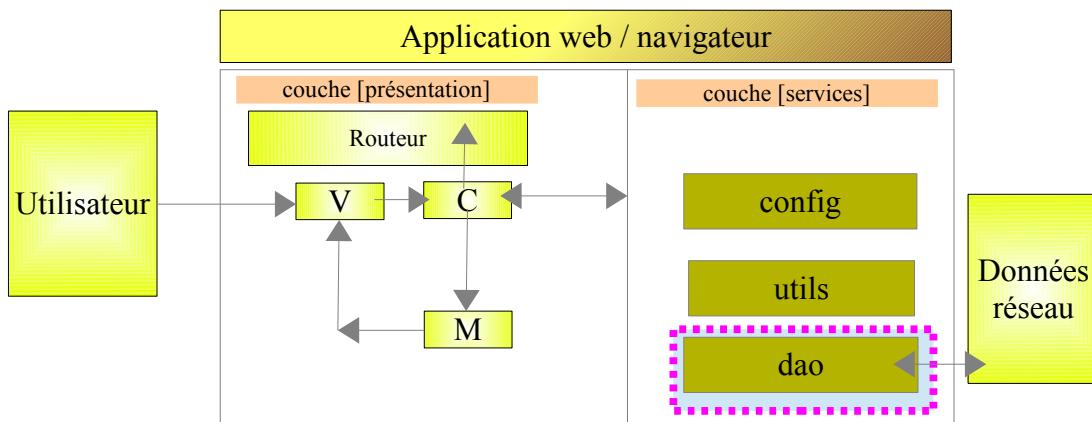
```
$scope.errors = { title: config.getMedecinsErrors, messages: utils.getErrors(result), show: true,
model: {}};
```

La liste des messages est fournie par la méthode [utils.getErrors] définie dans le service [utils]. Cette méthode est la suivante :

```
1. // analyse des erreurs dans la réponse du serveur JSON
2. function getErrors(data) {
3.   // data {err:n, messages:[], err!=0
4.   // erreurs
5.   var errors = [];
6.   // code d'erreur
7.   var err = data.err;
8.   switch (err) {
9.     case 2 :
10.       // not authorized
11.       errors.push('not_authorized');
12.       break;
13.     case 3 :
14.       // forbidden
15.       errors.push('forbidden');
16.       break;
17.     case 4 :
18.       // erreur locale
19.       errors.push('not_http_error');
20.       break;
21.     case 6 :
22.       // document non trouvé
23.       errors.push('not_found');
24.       break;
25.     default :
26.       // autres cas
27.       errors = data.messages;
28.       break;
29.
30.   }
31.   // si pas de msg, on en met un
32.   if (! errors || errors.length == 0) {
33.     errors=['error_unknown'];
34.   }
35.   // on rend la liste des erreurs
36.   return errors;
37. }
```

- lignes 2-3 : le paramètre [data] reçu est un objet avec deux attributs :
 - [err] : un code d'erreur ;
 - [messages] : une liste de messages ;
- ligne 5 : on va construire un tableau de messages d'erreur. Ces messages sont internationalisés. Pour cette raison, ce ne sont pas les messages eux-mêmes qu'on met dans le tableau, mais leurs clés d'internationalisation sauf à la ligne 27. Dans ce cas, on utilise l'attribut [messages] du paramètre [data]. Ces messages sont de vrais messages et non des clés de message. La vue V va cependant les traiter comme des clés de message qui ne seront alors pas trouvées. Dans ce cas, le module [translate] affiche la clé de message qu'il n'a pas trouvée, donc ici un vrai message. C'est le résultat souhaité ;
- lignes 32-34 : traitent le cas où [data.messages] ligne 27 vaut *null*. Cela arrive avec le service web écrit. Il aurait fallu éviter ce cas.

3.7.6.4 Le service [dao]



Le service [dao] assure les échanges HTTP avec le service web / JSON. Son code est le suivant :

```

1. angular.module("rdvmedecins")
2.   .factory('dao', ['$http', '$q', 'config', '$base64', 'utils',
3.     function ($http, $q, config, $base64, utils) {
4.
5.     // logs
6.     utils.debug("[dao] init");
7.
8.     // ----- méthodes privées
9.     // obtenir des données auprès du service web
10.    function getData(serverUrl, username, password, urlAction, info) {
11.        // opération asynchrone
12.        var task = $q.defer();
13.        // url requête HTTP
14.        var url = serverUrl + urlAction;
15.        // authentification basique
16.        var basic = "Basic " + $base64.encode(username + ":" + password);
17.        // la réponse
18.        var réponse;
19.        // les requêtes http doivent être toutes authentifiées
20.        var headers = $http.defaults.headers.common;
21.        headers.Authorization = basic;
22.        // on fait la requête HTTP
23.        var promise;
24.        if (info) {
25.            promise = $http.post(url, info, {timeout: config.timeout});
26.        } else {
27.            promise = $http.get(url, {timeout: config.timeout});
28.        }
29.        promise.then(success, failure);
30.        // on retourne la tâche elle-même afin qu'elle puisse être annulée
31.        return task;
32.

```

```

33.      // success
34.      function success(response) {
35.          // response.data={status:0, data:[med1, med2, ...]} ou {status:x, data:[msg1,
36.          msg2, ...]}
37.          utils.debug("[dao] getData[" + urlAction + "] success réponse", response);
38.          // réponse
39.          var payLoad = response.data;
40.          réponse = payLoad.status == 0 ? {err: 0, data: payLoad.data} : {err: 1, messages:
41.          payLoad.data};
42.          // on rend la réponse
43.          task.resolve(réponse);
44.      }
45.      // failure
46.      function failure(response) {
47.          utils.debug("[dao] getData[" + urlAction + "] error réponse", response);
48.          // on analyse le status
49.          var status = response.status;
50.          var error;
51.          switch (status) {
52.              case 401 :
53.                  // unauthorized
54.                  error = 2;
55.                  break;
56.              case 403:
57.                  // forbidden
58.                  error = 3;
59.                  break;
60.              case 404:
61.                  // not found
62.                  error = 6;
63.                  break;
64.              case 0:
65.                  // erreur locale
66.                  error = 4;
67.                  break;
68.              default:
69.                  // autre chose
70.                  error = 5;
71.          }
72.          // on rend la réponse
73.          task.resolve({err: error, messages: [response.statusText]});
74.      }
75.  }
76.  // ----- instance du service [dao]
77.  return {
78.     getData: getData
79.   }
80. }]);

```

- lignes 77-79 : le service n'a qu'un unique champ : la méthode [getData] qui permet d'obtenir des informations auprès du service web / JSON ;
- ligne 2 : apparaît une dépendance [\$http] que nous n'avions pas encore rencontrée. C'est un service prédéfini d'Angular qui permet le dialogue HTTP avec une entité distante ;
- ligne 6 : un log pour voir à quel moment de la vie de l'application, le code est exécuté ;
- ligne 10 : la méthode [getData] admet cinq paramètres :
 - [serverUrl] : l'URL racine du service web (<http://localhost:8080>) ;
 - [urlAction] : l'URL du service particulier demandé (/getAllMedecins) ;
 - [username] : le login de l'utilisateur ;
 - [password] : son mot de passe ;
 - [info] : objet rassemblant des informations complémentaires lorsque l'URL du service particulier demandé est demandé via une opération POST. Dans le cas de l'URL (/getAllMedecins), ce paramètre n'a pas été passé. Il est donc [undefined] ;
- ligne 12 : on crée une tâche asynchrone ;
- ligne 14 : l'URL complète du service demandé (<http://localhost:8080/getAllMedecins>) ;

- ligne 16 : l'authentification se fait en envoyant l'entête HTTP suivant :

```
Authorization:Basic code
```

où [code] est le code Base64 de la chaîne [username:password] ;

La ligne 16 construit la partie [Basic code] de l'entête HTTP ;

- ligne 18 : la réponse du service web ;
- ligne 20 : les entêtes HTTP envoyés par défaut par Angular dans une requête HTTP sont définis dans l'objet [\$http.defaults.headers.common]. L'entête [Authorization:Basic code] n'en fait pas partie ;
- ligne 21 : on l'ajoute aux entêtes HTTP à envoyer systématiquement. A gauche de l'affectation, on a l'entête [Authorization] à initialiser et à droite la valeur de l'entête, ici la valeur définie ligne 16. Ainsi si on écrit :

```
headers.Authorization = 'x';
```

Angular enverra l'entête HTTP :

```
Authorization : x
```

- ligne 23 : les méthodes du service [\$http] renvoient des promesses. Elles seront mémorisées dans la variable [promise] ;
- ligne 27 : parce qu'ici, le paramètre [info] a la valeur [undefined], c'est la ligne 27 qui est exécutée. L'URL (<http://localhost:8080/getAllMedecins>) est demandée avec un GET. Pour ne pas attendre trop longtemps, on fixe un délai d'attente maximum (timeout) pour obtenir la réponse du serveur. Par défaut, ce délai est d'une seconde ;
- ligne 29 : on définit les deux méthodes à exécuter lorsque la promesse est obtenue :
 - [success] : définie ligne 34, est la méthode à exécuter lorsque la promesse est obtenue sur un succès de la tâche ;
 - [failure] : définie ligne 45, est la méthode à exécuter lorsque la promesse est obtenue sur un échec de la tâche ;
 - les deux méthodes (on devrait dire fonctions) sont définies à l'intérieur de la fonction [getData]. C'est possible en Javascript. Les variables définies dans [getData] sont connues dans les deux fonctions internes [success, failure] ;
- ligne 31 : on retourne la tâche créée ligne 12. Il faut se rappeler ici le code appelant :

```
1.     promise = promise.then(function () {
2.         // on demande la liste des médecins;
3.         task = dao.getData($scope.server.url, $scope.server.login, $scope.server.password,
4.             config.urlSrvMedecins);
5.         return task.promise;
});
```

Ligne 3 ci-dessus, on récupère bien une tâche.

- ligne 34 : la fonction [success] est exécutée plus tard dans le temps, lorsque l'appel HTTP se termine avec succès. Cette notion de succès est liée à la première ligne d'une réponse HTTP. Celle-ci a la forme :

```
HTTP/1.1 code texte
```

Le code est un texte de trois chiffres qui indique si l'appel a réussi ou non. Grossièrement, on peut dire que les codes 2xx et 3xx sont des codes de réussite, les autres étant des codes d'échec. Le texte est un court texte d'explication. Voici deux réponses possibles, l'un en cas de réussite, l'autre en cas d'échec :

```
HTTP/1.1 200 OK
```

```
HTTP/1.1 404 Not Found
```

- ligne 36 : on affiche sur la console la réponse du serveur. Dans l'erreur [404 Not Found], on obtient quelque chose comme :

```
[dao] getData[/getAllMedecins] error réponse : {"data":"...","status":404,"config":{},"statusText":"Not Found"}
```

Dans cette réponse, nous n'utiliserons que les champs [data], [status] et [statusText].

- ligne 38 : on récupère le champ [data] de la réponse. Il aura l'une des formes suivantes :
 - {status: 0, data: [med1, med2, ...]} où [medi] est un objet représentant un médecin (titre, prenom, nom),
 - {status: n, data: [msg1, msg2, ...]} où [msgi] est un message d'erreur et n est différent de 0 ;

```

localhost:8080/getAllMedecins
{
  "status": 0,
  "data": [
    {"prenom": "Marie", "id": 1, "nom": "PELISSIER", "titre": "Mme"}, 
    {"prenom": "Jacques", "id": 2, "nom": "BROMARD", "titre": "Mr"}, 
    {"prenom": "Philippe", "id": 3, "nom": "JANDOT", "titre": "Mr"}, 
    {"prenom": "Justine", "id": 4, "nom": "JACQUEMOT", "titre": "Melle"}
  ]
}

localhost:8080/getAllMedecins
{
  "status": -1,
  "data": [
    "Could not open JPA EntityManager for transaction; nested exception is javax.persistence.PersistenceException: org.hibernate.exception.GenericJDBCException: Could not open connection",
    "org.hibernate.exception.GenericJDBCException: Could not open connection",
    "org.hibernate.exception.GenericJDBCException: Could not open connection",
    "Cannot create PoolableConnectionFactory (Communications link failure\n\nThe last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.)",
    "Communications link failure\n\nThe last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.",
    "Connection refused: connect"
  ]
}

```

- ligne 39 : on construit la réponse {0,data} ou {n,messages}. La première réponse contient les médecins dans le champ [data]. La seconde signale une erreur qui s'est produite côté serveur. Celui-ci a généré celle-ci, généré un code d'erreur dans [err] et une liste de messages d'erreur dans [data]. Dans les deux cas, il renvoie un code HTTP 200 indiquant que l'ordre HTTP a été traité complètement. C'est pour cela que les deux cas sont traités dans la même fonction [success] ;
- ligne 41 : la tâche est terminée [task.resolve] et on rend l'une des deux réponses :
 - {err: 0, data: [med1, med2, ...]} où [med1] est un objet représentant un médecin (titre, prenom, nom),
 - {err: n, messages: [msg1, msg2, ...]} où [msg1] est un message d'erreur et n est différent de 0 ;

Il faut relier ce code à la façon dont cette réponse est récupérée dans le code appelant du contrôleur :

```

1.   // on analyse le résultat de l'appel précédent
2.   promise.then(function (result) {
3.     // result={err: 0, data: [med1, med2, ...]}
4.     // result={err: n, messages: [msg1, msg2, ...]}
5.     ...
6.   }

```

La réponse de [task.resolve(réponse)] se retrouve ci-dessus dans la variable [result].

- ligne 45 : la fonction [failure] lorsque la tâche asynchrone se termine sur un échec. Il y a deux cas possibles :
 - le serveur signale cet échec en renvoyant un code qui n'est ni 2xx ni 3xx,
 - Angular annule l'appel HTTP. Il n'y a alors pas d'appel. Il y a une exception Angular mais pas de code d'erreur HTTP renvoyé par le serveur. C'est par exemple le cas, si on fournit une URL invalide qui ne peut être appellée ;
- ligne 46 : on affiche la réponse sur la console ;
- ligne 48 : on se rappelle que la réponse du serveur a la forme :

```
{"data":"...","status":404,"config":{...}, "statusText":"Not Found"}
```

Ligne 48, on récupère l'attribut [status] ci-dessus ;

- lignes 50-70 : à partir du code d'erreur HTTP, on va générer un nouveau code d'erreur pour cacher aux codes appelants, la nature HTTP de la méthode [daoGetData]. On peut vérifier que dans le contrôleur qui utilise cette méthode, rien ne laisse supposer qu'il y a un appel HTTP dans la méthode ;
 - ligne 51 : l'erreur [401] correspond à un échec de l'authentification (mot de passe incorrect par exemple),

- ligne 55 : l'erreur [403] correspond à un appel non autorisé. L'utilisateur s'est correctement authentifié mais il n'a pas les droits suffisants pour demander l'URL qu'il a demandée. Cela arrivera avec l'utilisateur [user / user]. Celui-ci existe bien en base de données mais n'a pas le droit d'utiliser l'application. Seul l'utilisateur [admin / admin] a ce droit ;
- ligne 59 : l'erreur [404] correspond à une URL non trouvée. L'erreur peut avoir plusieurs causes :
 - l'utilisateur a fait une erreur de saisie dans l'URL du service ;
 - le service web n'a pas été lancé ;
 - le service web n'a pas répondu assez vite (délai d'une seconde par défaut) ;
- ligne 63 : le code d'erreur HTTP 0 n'existe pas. On est dans le cas où Angular n'a pas fait l'appel HTTP demandé parce que l'URL saisie par l'utilisateur est invalide et ne peut être appelée. Nous allons par la suite rencontrer d'autres cas où Angular est amené à ne pas exécuter l'appel HTTP demandé ;
- ligne 72 : on termine la tâche avec succès (task.resolve) en renvoyant une réponse du type `{err, messages}` où le tableau `[messages]` n'est formé que du seul message `[response.statusText]`. Dans le cas où Angular n'a pas fait l'appel HTTP demandé, on aura une chaîne vide ;

Maintenant qu'on a une vue à la fois globale et détaillée de l'application, nous pouvons commencer les tests.

3.7.6.5 Tests de l'application - 1

Commençons avec des saisies valides :

Rdvmedecins - v1

The screenshot shows the application's main interface with the title "Rdvmedecins - v1". It contains four input fields: "Temps d'attente : 0", "URL du service web : http://localhost:8080", "Login : admin", and "Mot de passe :". Below these fields is a blue button labeled "Liste des médecins" with a cursor icon pointing to it.

Rdvmedecins - v1

The screenshot shows the application's main interface with the title "Rdvmedecins - v1". It contains the same four input fields as the previous screenshot. Below the "Liste des médecins" button is a red error message box containing the text: "Les erreurs suivantes se sont produites lors du chargement de la liste des médecins" followed by a bulleted list: "• L'accès au serveur n'a pu être réalisé. Vérifiez sa disponibilité". A yellow box labeled "2" is overlaid on the bottom right corner of the error message box.

- en [1], on met 0 pour ne pas avoir d'attente ;
- en [2], on a un message d'erreur alors que les saisies sont correctes. Nous n'avons pas présenté les différents messages d'erreur. Celui affiché en [2] est un message générique associé à l'erreur 0 qui correspond à une exception Angular. Angular a rencontré un problème qui l'a empêché de faire un appel HTTP. Dans ces cas là, il faut regarder les logs de la console Javascript. Il y a deux façons de faire cela :
 - faire [F12] dans le navigateur Chrome ;

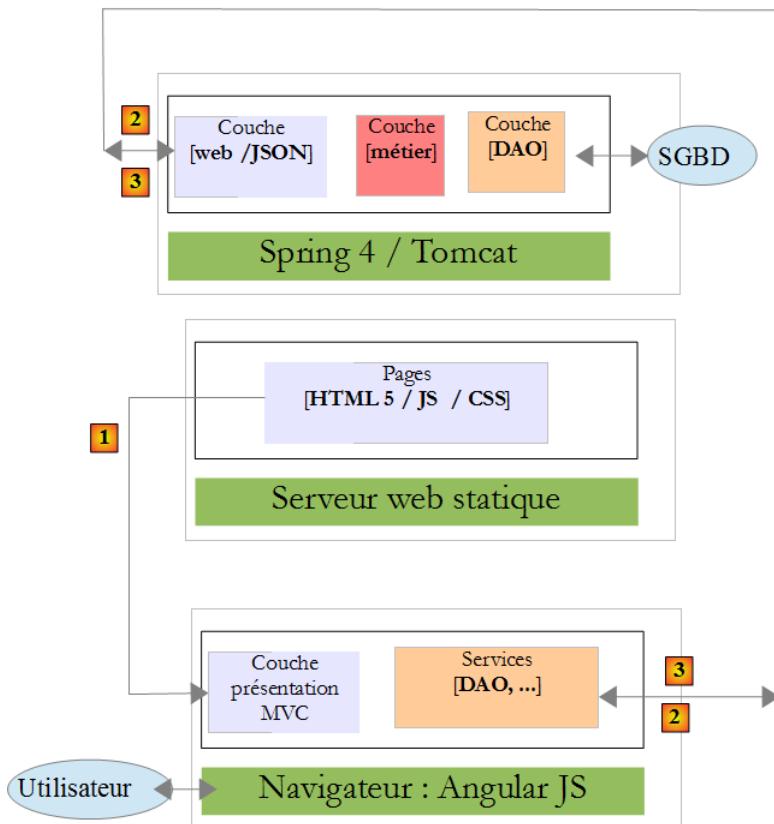
- utiliser la console de Webstorm ;

Dans la console de Webstorm, nous trouvons divers messages dont celui-ci :

```
1. XMLHttpRequest cannot load http://localhost:8080/getAllMedecins. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:63342' is therefore not allowed access.
2. [dao] getData[/getAllMedecins] error réponse : {"data":"","status":0,"config":{"method":"GET","transformRequest":[null],"transformResponse":[null],"timeout":1000,"url":"http://localhost:8080/getAllMedecins","headers":{"Accept":"application/json, text/plain, */*","Authorization":"Basic YWRtaW46YWRtaW4="}),"statusText":""}
```

- ligne 1 : Angular signale une erreur sur laquelle nous allons revenir ;
- ligne 2 : le log de la méthode [dao.getData]. On y trouve des choses intéressantes :
 - [status] vaut 0, indiquant par là, qu'il n'y a pas eu d'appel HTTP. En conséquence [statusText] est vide,
 - [url] vaut [http://localhost:8080/getAllMedecins] ce qui est correct ;
 - l'entête HTTP d'authentification [Authorization]:"Basic YWRtaW46YWRtaW4="] est lui aussi correct ;

Bon alors pourquoi ça n'a pas marché ? La phrase clé des logs est [No 'Access-Control-Allow-Origin' header is present]. Pour la comprendre, il faut faire une longue explication. Commençons par revenir sur l'architecture générale de l'application client / serveur :



- les pages HTML / CSS / JS de l'application Angular viennent du serveur [1] ;
- en [2], le service [dao] fait une requête à un autre serveur, le serveur [2]. Et bien, ça c'est interdit par le navigateur qui exécute l'application Angular parce que c'est une faille de sécurité. L'application ne peut interroger que le serveur d'où elle vient, ç-à-d le serveur [1] ;

En fait, il est inexact de dire que le navigateur interdit à l'application Angular d'interroger le serveur [2]. Elle l'interroge en fait pour lui demander s'il autorise un client qui ne vient pas de chez lui à l'interroger. On appelle cette technique de partage, le **CORS**

(Cross-Origin Resource Sharing). Le serveur [2] donne son accord en envoyant des entêtes HTTP précis. C'est parce qu'ici, notre serveur [2] ne les a pas envoyées que le navigateur a refusé de faire l'appel HTTP demandé par l'application.

Entrons maintenant dans les détails. Examinons les échanges réseau qui ont eu lieu lors de l'appel HTTP. Pour cela, dans le navigateur Chrome, nous faisons [F12] pour obtenir les outils du développeur et nous sélectionnons l'onglet [Network] pour voir les échanges réseau :

The screenshot shows the Network tab of the Chrome DevTools. A red box labeled '1' highlights the 'Network' tab. To its right, a blue box labeled '2' highlights the 'Liste des médecins' button on a web page titled 'Rdvmedecins - v1'. The page contains fields for 'Temps d'attente : 0', 'URL du service web : http://localhost:8080', 'Login : admin', 'Mot de passe :', and a 'Liste des médecins' button.

- en [1], nous sélectionnons l'onglet [network] ;
- en [2], nous demandons la liste des médecins ;

Nous obtenons les informations suivantes dans l'onglet [network] :

The screenshot shows the Network tab of the Chrome DevTools. A red box labeled '1' highlights the 'Request Headers' section for the 'getAllMedecins' request. It lists headers such as Remote Address, Request URL, Request Method, Status Code, and various header fields like Connection, Host, and Origin. A blue box labeled '2' highlights the 'Response Headers' section for the same request. It lists standard HTTP headers like Content-Type, Content-Length, and Date, along with specific CORS-related headers like Access-Control-Allow-Origin and Access-Control-Allow-Methods.

- en [1], les informations envoyées au serveur ;
- en [2], la réponse de celui-ci ;

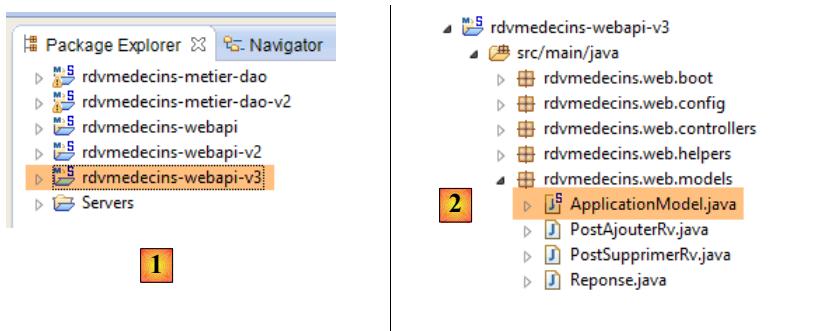
On peut voir dans [1], que le navigateur a envoyé une requête HTTP [OPTIONS] sur l'URL demandée. [OPTIONS] est une des commandes HTTP possibles avec [GET] et [POST] plus connues. Elle permet de demander des informations à un serveur notamment sur les options HTTP qu'il supporte, d'où le nom de la commande. Le serveur fait sa réponse en [2]. Pour indiquer qu'il accepte des requêtes de clients qui ne sont pas dans son domaine, il doit renvoyer un entête particulier appelé [Access-Control-Allow-Origin]. Et c'est parce qu'il ne l'a pas renvoyé qu'Angular n'a pas exécuté l'appel HTTP demandé et a renvoyé l'erreur :

XMLHttpRequest cannot load http://localhost:8080/getAllMedecins. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:63342' is therefore not allowed access.

Nous devons donc modifier notre serveur pour qu'il envoie l'entête HTTP attendu.

3.7.6.6 Modification du serveur web / JSON

Nous revenons sous Eclipse. Afin de conserver l'acquis, nous dupliquons la version actuelle du serveur web / JSON [rdvmedecins-webapi-v2] dans [rdvmedecins-webapi-v3] [1] :

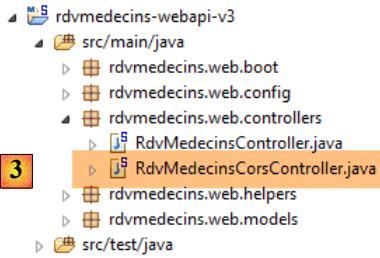


Nous faisons une première modification dans [ApplicationModel] qui est l'un des éléments de configuration du service web :

```
1. package rdvmedecins.web.models;
2.
3. ...
4.
5. @Component
6. public class ApplicationModel implements IMetier {
7.
8.     // la couche [métier]
9.     @Autowired
10.    private IMetier métier;
11.
12.    // données provenant de la couche [métier]
13.    private List<Medecin> médecins;
14.    private List<Client> clients;
15.    private List<String> messages;
16.    // données de configuration
17.    private boolean CORSneeded = true;
18.
19. ...
20.
21.    public boolean isCORSneeded() {
22.        return CORSneeded;
23.    }
24.
25. }
```

- ligne 17 : nous créons un booléen qui indique si on accepte ou non les clients étrangers au domaine du serveur ;
- lignes 21-23 : la méthode d'accès à cette information ;

Puis nous créons un nouveau contrôleur Spring MVC [3] :



La classe [RdvMedecinsCorsController] est la suivante :

```

1. package rdvmedecins.web.controllers;
2.
3. import javax.servlet.http.HttpServletResponse;
4.
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.stereotype.Controller;
7. import org.springframework.web.bind.annotation.RequestMapping;
8. import org.springframework.web.bind.annotation.RequestMethod;
9.
10. import rdvmedecins.web.models.ApplicationModel;
11.
12. @Controller
13. public class RdvMedecinsCorsController {
14.
15.     @Autowired
16.     private ApplicationModel application;
17.
18.     // envoi des options au client
19.     private void sendOptions(HttpServletResponse response) {
20.         if (application.isCORSneeded()) {
21.             // on fixe le header CORS
22.             response.addHeader("Access-Control-Allow-Origin", "*");
23.         }
24.     }
25. }
26.
27. // liste des médecins
28. @RequestMapping(value = "/getAllMedecins", method = RequestMethod.OPTIONS)
29. public void getAllMedecins(HttpServletRequest response) {
30.     sendOptions(response);
31. }
32. }
```

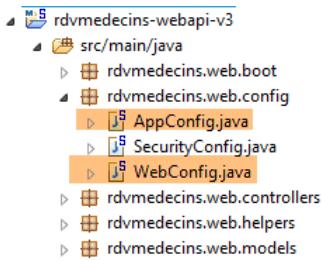
- lignes 28-31 : définissent un contrôleur pour l'URL [/getAllMedecins] lorsqu'elle est demandée avec la commande HTTP [OPTIONS] ;
- ligne 29 : la méthode [getAllMedecins] admet pour paramètre l'objet [HttpServletRequest] qui va être envoyé au client qui a fait la demande. Cet objet est injecté par Spring ;
- ligne 30 : on délègue le traitement de la demande à la méthode privée des lignes 19-25 ;
- lignes 15-16 : l'objet [ApplicationModel] est injecté ;
- lignes 20-23 : si le serveur est configuré pour accepter les clients étrangers à son domaine, alors on envoie l'en-tête HTTP :

`Access-Control-Allow-Origin: *`

qui signifie que le serveur accepte les clients de tout domaine (*).

Nous sommes désormais prêts pour de nouveaux tests. Nous lançons la nouvelle version du service web et nous découvrons que le problème reste entier. Rien n'a changé. Si ligne 30 ci-dessus, on met un affichage console, celui-ci n'est jamais affiché montrant par là que la méthode [getAllMedecins] de la ligne 29 n'est jamais appelée.

Après quelques recherches, on découvre que Spring MVC traite lui-même les commandes HTTP [OPTIONS] avec un traitement par défaut. Aussi c'est toujours Spring qui répond et jamais la méthode [getAllMedecins] de la ligne 29. Ce comportement par défaut de Spring MVC peut être changé. Nous introduisons une nouvelle classe de configuration, pour configurer le nouveau comportement :



La nouvelle classe de configuration [WebConfig] est la suivante :

```

1. package rdvmedecins.web.config;
2.
3. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
4. import org.springframework.context.annotation.Bean;
5. import org.springframework.web.servlet.DispatcherServlet;
6. import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
7.
8. @Configuration
9. public class WebConfig extends WebMvcConfigurerAdapter {
10.
11.     // configuration dispatcherservlet pour les headers CORS
12.     @Bean
13.     public DispatcherServlet dispatcherServlet() {
14.         DispatcherServlet servlet = new DispatcherServlet();
15.         servlet.setDispatchOptionsRequest(true);
16.         return servlet;
17.     }
18. }
```

- ligne 8 : la classe est une classe de configuration Spring. Elle déclare des beans qui seront placés dans le contexte de Spring ;
- ligne 12 : le bean [dispatcherServlet] sert à définir la servlet qui gère les demandes des clients. Elle est de type [DispatcherServlet]. Cette servlet est normalement créée par défaut. Si on la crée nous-mêmes, on peut alors la configurer ;
- ligne 14 : on crée une instance de type [DispatcherServlet] ;
- ligne 15 : on demande à ce que la servlet fasse suivre à l'application les commandes HTTP [OPTIONS] ;
- ligne 16 : on rend la servlet ainsi configurée ;

Il nous reste à modifier la classe [AppConfig] :

```

1. package rdvmedecins.web.config;
2.
3. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Import;
6.
7. import rdvmedecins.config.DomainAndPersistenceConfig;
8.
9. @EnableAutoConfiguration
10. @ComponentScan(basePackages = { "rdvmedecins.web" })
11. @Import({ DomainAndPersistenceConfig.class, SecurityConfig.class, WebConfig.class })
12. public class AppConfig {
13.
14. }
```

- ligne 11 : la nouvelle classe de configuration [WebConfig] est importée ;

3.7.6.7 Tests de l'application - 2

Nous lançons la nouvelle version du service web / JSON et essayons d'obtenir la liste des médecins avec notre client Angular. Nous examinons les échanges réseau dans l'onglet [Network] :

The screenshot shows the Network tab in the Chrome DevTools. A request for 'getAllMedecins' is selected. In the Headers section, the 'Access-Control-Allow-Origin' header is highlighted with a yellow box labeled '1'. In the Response Headers section, the 'Access-Control-Allow-Headers' header is highlighted with a yellow box labeled '2'.

- en [1], on peut constater que l'entête HTTP [`Access-Control-Allow-Origin: *`] est désormais présent dans la réponse du serveur. Et pourtant ça ne marche toujours pas. Nous examinons en [2], les logs de la console. On y trouve le log suivant :

```
XMLHttpRequest cannot load http://localhost:8080/getAllMedecins. Request header field Authorization is not allowed by Access-Control-Allow-Headers
```

On voit que le navigateur attend un nouvel entête HTTP [`Access-Control-Allow-Headers`] qui lui dirait qu'on a le droit de lui envoyer l'entête d'authentification :

```
Authorization:Basic code
```

Cela peut être bon signe. Angular a peut être voulu envoyer la commande HTTP GET. Mais comme celle-ci est accompagnée d'un entête d'authentification, il demande si le serveur accepte celui-ci.

Nous modifions notre serveur web / JSON pour envoyer cet entête. La classe [`RdvMedecinsCorsController`] évolue comme suit :

```
1.     // envoi des options au client
2.     private void sendOptions(HttpServletRequest response) {
3.         if (application.isCORSneeded()) {
4.             // on fixe le header CORS
5.             response.addHeader("Access-Control-Allow-Origin", "*");
6.             // on autorise le header [Authorization]
7.             response.addHeader("Access-Control-Allow-Headers", "Authorization");
8.     }
```

- les lignes 6-7 ajoutent l'entête manquant.

Nous relançons le serveur et redemandons la liste des médecins avec le client Angular :

Rdvmedecins - v1

Temps d'attente : 0

URL du service web : http://localhost:8080

Login : admin

Mot de passe : *****

Liste des médecins

Liste des médecins

- Mme Marie PELISSIER
- Mr Jacques BROMARD
- Mr Philippe JANDOT
- Melle Justine JACQUEMOT

Cette fois, c'est bon. Les logs console montrent la réponse reçue par la méthode [dao.getData] :

```
1. [dao] getData[/getAllMedecins] success réponse : {"data": {"status": 0, "data": [{"id": 1, "version": 1, "titre": "Mme", "nom": "PELISSIER", "prenom": "Marie"}, {"id": 2, "version": 1, "titre": "Mr", "nom": "BROMARD", "prenom": "Jacques"}, {"id": 3, "version": 1, "titre": "Mr", "nom": "JANDOT", "prenom": "Philippe"}, {"id": 4, "version": 1, "titre": "Melle", "nom": "JACQUEMOT", "prenom": "Justine"}]}, "status": 200, "config": {"method": "GET", "transformRequest": [null], "transformResponse": [null], "timeout": 1000, "url": "http://localhost:8080/getAllMedecins", "headers": {"Accept": "application/json, text/plain, */*", "Authorization": "Basic YWRtaW46YWRtaW4="}}, "statusText": "OK"}
```

On voit que :

- le serveur a renvoyé un code d'erreur [status=200] avec le message [statusText=OK]. C'est pourquoi on est dans la fonction [success] ;
- le serveur a renvoyé un objet [data] avec deux champs :
 - [status] : (à ne pas confondre avec le code d'erreur HTTP [status]). Ici [status=0] indique que l'URL [/getAllMedecins] a été traitée sans erreur ;
 - [data] : qui contient la liste JSON des médecins ;

Montrons maintenant d'autres cas intéressants :

On se trompe dans les identifiants [login, password] :

Rdvmedecins - v1

Temps d'attente :

URL du service web :

Login :

Mot de passe :

Les erreurs suivantes se sont produites lors du chargement de la liste des médecins

- Erreur d'authentification

On se connecte sous l'identité [user / user] qui n'a pas accès à l'application (seul [admin] y a accès) :

Rdvmedecins - v1

Temps d'attente :

URL du service web :

Login :

Mot de passe :

Les erreurs suivantes se sont produites lors du chargement de la liste des médecins

- Accès refusé

Cette fois-ci, l'erreur n'est plus [Erreur d'authentification] mais [Accès refusé].

3.7.7 Exemple 7 : liste des clients

Nous reprenons l'application précédente pour cette fois présenter la liste des clients dans une liste déroulante de type [Bootstrap select] (cf paragraphe 3.6.6, page 151).

3.7.7.1 La vue V

La vue initiale sera la suivante :

Rdvmedecins - v1

The screenshot shows a simple web application interface. At the top, there are four input fields: 'Temps d'attente' with value '0', 'URL du service web' with value 'http://localhost:8080', 'Login' with value 'admin', and 'Mot de passe' with value '*****'. Below these fields is a blue rectangular button with the text 'Liste des clients'.

Pour obtenir la vue V, nous dupliquons le code [app-16.html] dans [app-17.html] et le modifions de la façon suivante :

```
1. <div class="container" >
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <!-- le message d'attente -->
5.   <div class="alert alert-warning" ng-show="waiting.visible" >
6.   ...
7.   </div>
8.
9.   <!-- la demande -->
10.  <div class="alert alert-info" ng-hide="waiting.visible" >
11.  ...
12.    <button class="btn btn-primary" ng-click="execute()">{{clients.title|translate}}</button>
13.  </div>
14.
15.  <!-- la liste des clients -->
16.  <div class="row" style="margin-top: 20px" ng-show="clients.show">
17.    <div class="col-md-3">
18.      <h2 translate="{{clients.title}}></h2>
19.      <select data-style="btn-primary" class="selectpicker">
20.        <option ng-repeat="client in clients.data" value="{{client.id}}>
21.          {{client.titre}} {{client.prenom}} {{client.nom}}
22.        </option>
23.      </select>
24.    </div>
25.  </div>
26.
27.  <!-- la liste d'erreurs -->
28.  <div class="alert alert-danger" ng-show="errors.show">
29.  ...
30.  </div>
31.
32. </div>
33. ....
34. <script type="text/javascript" src="rdvmedecins-05.js"></script>
```

- lignes 5-7 : le bandeau d'attente ne change pas ;
- lignes 10-13 : le formulaire ne change pas, si ce n'est le libellé du bouton (ligne 12) ;
- lignes 28-30 : le bandeau des erreurs ne change pas ;
- lignes 16-25 : l'affichage des clients se fait dans une liste déroulante stylée par le composant [Bootstrap-selectpicker] (attributs **data-style**, **class**, ligne 19) ;
- ligne 20 : on utilise la directive [ng-repeat] pour générer les différentes options de la liste déroulante. On notera que le **libellé** d'une option est de type [Mme Julianne Tatou] et que la valeur de l'option est de type [100] où 100 est l'identifiant **id** du client affiché ;
- ligne 34 : le code Javascript migre dans un nouveau fichier [[rdvmedecins-05](#)] ;

3.7.7.2 Le contrôleur C et le modèle M

Le code Javascript du fichier [[rdvmedecins-05](#)] est obtenu par recopie du fichier [[rdvmedecins-04](#)] :

```

4   // ----- module Angular
5   angular.module("rdvmedecins", [
6     "pascalprecht.translate",
7     "base64"
8   ]);
9   // configuration initiale
10  angular.module("rdvmedecins")
11  .config(['$translateProvider', function ($translateProvider) {...}]);
12
13  // services
14  angular.module("rdvmedecins")
15  .factory('config', function () {...})
16;
17
18  angular.module("rdvmedecins")
19  .factory('utils', ['$config', '$timeout', '$q', function ($config, $timeout, $q) {...}]);
20
21  angular.module("rdvmedecins")
22  .factory('dao', [...]);
23
24  // contrôleur
25  angular.module("rdvmedecins")
26  .controller('rdvMedecinsCtrl', [...]);
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

```

Quasiment rien ne change, sauf dans le contrôleur qui est désormais adapté pour fournir la liste des clients :

```

1. angular.module("rdvmedecins")
2.   .controller('rdvMedecinsCtrl', ['$scope', 'utils', 'config', 'dao', '$translate',
3.     function ($scope, utils, config, dao, $translate) {
4.       // ----- initialisation modèle
5.       // modèle
6.       $scope.waiting = {text: config.msgWaiting, visible: false, cancel: cancel, time: undefined};
7.       $scope.waitingTimeText = config.waitingTimeText;
8.       $scope.server = {url: undefined, login: undefined, password: undefined};
9.       $scope.clients = {title: config.listClients, show: false, model: {}};
10.      $scope.errors = {show: false, model: {}};
11.      $scope.urlServerLabel = config.urlServerLabel;
12.      $scope.loginLabel = config.loginLabel;
13.      $scope.passwordLabel = config.passwordLabel;
14.
15.      // tâche asynchrone
16.      var task;
17.
18.      // exécution action
19.      $scope.execute = function () {
20.        // on met à jour l'UI
21.        $scope.waiting.visible = true;
22.        $scope.clients.show = false;
23.        $scope.errors.show = false;
24.        // attente simulée
25.        task = utils.waitForSomeTime($scope.waiting.time);
26.        var promise = task.promise;
27.        // attente
28.        promise = promise.then(function () {
29.          // on demande la liste des clients;
30.          task = dao.getData($scope.server.url, $scope.server.login, $scope.server.password,
config.urlSrvClients);
31.          return task.promise;
32.        });
33.        // on analyse le résultat de l'appel précédent
34.        promise.then(function (result) {
35.          // result={err: 0, data: [client1, client2, ...]}
36.          // result={err: n, messages: [msg1, msg2, ...]}
37.          if (result.err == 0) {
38.            // on met les données acquises dans le modèle
39.            $scope.clients.data = result.data;
40.            // on met à jour l'UI
41.            $scope.clients.show = true;
42.            $scope.waiting.visible = false;
43.            // on style la liste déroulante

```

```

44.      $('.selectpicker').selectpicker();
45.    } else {
46.      // il y a eu des erreurs pour obtenir la liste des clients
47.      $scope.errors = { title: config.getClientsErrors, messages: utils.getErrors(result), show:
48.        true, model: {}};
48.      // on met à jour l'UI
49.      $scope.waiting.visible = false;
50.    }
51.  });
52. };
53.
54. // annulation attente
55. function cancel() {
56.   // on termine la tâche
57.   task.reject();
58.   // on met à jour l'UI
59.   $scope.waiting.visible = false;
60.   $scope.clients.show = false;
61.   $scope.errors.show = false;
62. }
63. ]
64. ])
65. ;

```

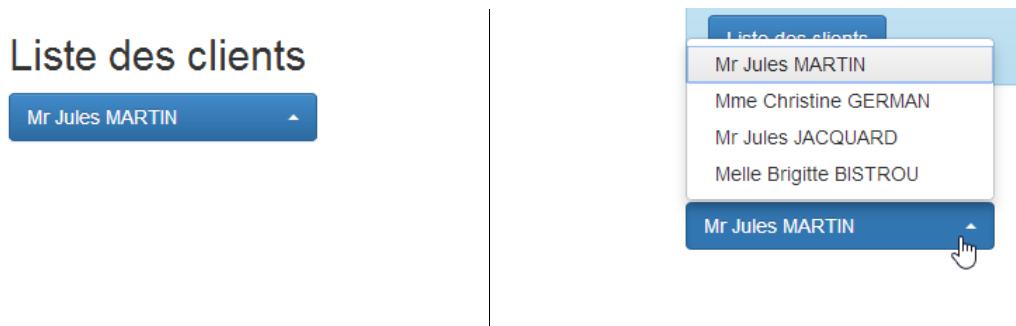
- très peu de choses changent dans le contrôleur. Il fournissait une liste de médecins. Il fournit désormais une liste de clients ;
- ligne 9 : `[$scope.clients]` sera le modèle du bandeau des clients dans la vue V ;
- ligne 30 : c'est l'URL `[/getAllClients]` qui est désormais utilisée ;
- lignes 35-36 : les deux formes de réponse rendue par la méthode `[dao.getData]`. On a maintenant des clients au lieu de médecins ;
- ligne 44 : une instruction assez rare dans un code Angular. On manipule directement le DOM (Document Object Model). Ici on veut appliquer la méthode `[selectpicker]` (fait partie de `[bootstrap-select.min.js]`) aux éléments du DOM qui ont la classe `[selectpicker]` `[$('.selectpicker')]`. Il n'y en a qu'un, la liste déroulante :

```

<select data-style="btn-primary" class="selectpicker" select-enable="">
.....
</select>

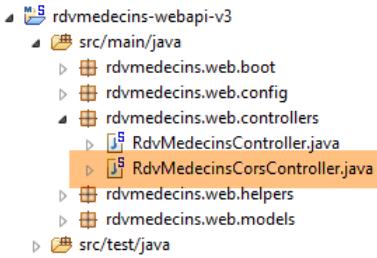
```

Au paragraphe 3.6.6, page 151, il a été montré que cela stylisait la liste déroulante de la façon suivante :



Comme il a été fait pour les médecins, nous sommes amenés à modifier le service web également.

3.7.7.3 Modification du service web - 1



La classe [RdvMedecinsController] s'enrichit d'une nouvelle méthode :

```

1. package rdvmedecins.web.controllers;
2.
3. ...
4.
5. @Controller
6. public class RdvMedecinsCorsController {
7.
8.     @Autowired
9.     private ApplicationModel application;
10.
11.    // envoi des options au client
12.    private void sendOptions(HttpServletRequest response) {
13.        if (application.isCORSneeded()) {
14.            // on fixe le header CORS
15.            response.addHeader("Access-Control-Allow-Origin", "*");
16.            // on autorise le header [Authorization]
17.            response.addHeader("Access-Control-Allow-Headers", "Authorization");
18.        }
19.    }
20. }
21.
22. // liste des médecins
23. @RequestMapping(value = "/getAllMedecins", method = RequestMethod.OPTIONS)
24. public void getAllMedecins(HttpServletRequest response) {
25.     sendOptions(response);
26. }
27.
28. // liste des clients
29. @RequestMapping(value = "/getAllClients", method = RequestMethod.OPTIONS)
30. public void getAllClients(HttpServletRequest response) {
31.     sendOptions(response);
32. }
33. }
```

- lignes 29-32 : la méthode [getAllClients] va gérer la demande HTTP [OPTIONS] que va lui envoyer le navigateur ;

3.7.7.4 Tests de l'application – 1

Nous sommes désormais prêts pour un test. Nous lançons le serveur web puis entrons des valeurs valides dans le formulaire Angular. Nous obtenons la réponse suivante :

Rdvmedecins - v1

Temps d'attente : 0

URL du service web : <http://localhost:8080>

Login : admin

Mot de passe :

Liste des clients

Les erreurs suivantes se sont produites lors du chargement de la liste des clients
• L'accès au serveur n'a pu être réalisé. Vérifiez sa disponibilité

Ce message d'erreur est affiché lorsqu'Angular n'a pu faire la requête HTTP demandée. Il faut en chercher alors les causes dans les logs de la console. On y trouve le message suivant :

```
XMLHttpRequest cannot load http://localhost:8080/getAllClients. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:63342' is therefore not allowed access.
```

Un problème qu'on croyait résolu. On va alors voir les échanges réseau qui se sont produits :

Elements Network Sources Timeline Profiles Resources Audits Console AngularJS		
Name	Method	Status
Path		Text
getAllClients	OPTIONS	200 OK
getAllClients	GET	(canceled)

On voit que l'opération [getAllClients] avec la méthode HTTP [OPTIONS] s'est bien passée mais que l'opération [getAllClients] avec la méthode HTTP [GET] a été annulée. La réponse à la demande [OPTIONS] a été la suivante :

```
▼ Response Headers view parsed
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Authorization
Allow: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, PATCH
Content-Length: 0
Date: Wed, 25 Jun 2014 09:48:56 GMT
```

Les entêtes HTTP du CORS sont bien là. Examinons maintenant les échanges HTTP lors du GET :

```

: Headers Preview Response
Request URL: http://localhost:8080/getAllClients
▼ Request Headers CAUTION: Provisional headers are shown.
  Accept: application/json, text/plain, /*
  Authorization: Basic YWRtaW46YWRtaW4=
  Cache-Control: no-cache
  Origin: http://localhost:63342
  Pragma: no-cache
  Referer: http://localhost:63342/rdvmedecins-angular-v1/app-17.html
  User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (
```

La requête HTTP semble correcte. On voit notamment l'entête d'authentification.

Outre le message d'erreur précédent, on trouve dans les logs console, le message suivant :

```
[dao] getData[/getAllClients] error réponse : {"data": "", "status": 0, "config": {"method": "GET", "transformRequest": [null], "transformResponse": [null], "timeout": 1000, "url": "http://localhost:8080/getAllClients", "headers": {"Accept": "application/json, text/plain, /*", "Authorization": "Basic YWRtaW46YWRtaW4="}}, "statusText": ""}
```

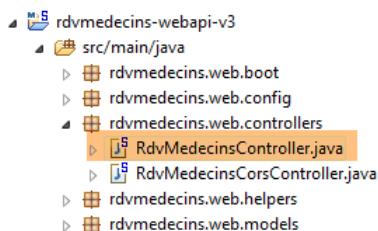
C'est le log que fait systématiquement la méthode [dao.getData] à la réception de la réponse à sa demande HTTP. On peut remarquer deux choses :

- [status=0] : cela veut dire que c'est Angular qui a annulé la requête HTTP ;
- [method=GET] : et c'est la requête GET qui a été annulée ;

Mis bout à bout avec le premier message, cela veut dire que pour la requête GET également, Angular attend ici des entêtes CORS. Or actuellement, notre service web ne les envoie que pour la requête HTTP [OPTIONS]. Il est très étrange de rencontrer cette erreur maintenant et pas pour la liste des médecins. Je n'ai pas d'explications.

Il faut donc modifier de nouveau le service web.

3.7.7.5 Modification du service web – 2



Les méthodes [GET] et [POST] sont traitées dans la classe [RdvMedecinsController]. Nous devons la modifier pour que ces méthodes envoient les entêtes CORS. Nous le faisons de la façon suivante :

```

1. @RestController
2. public class RdvMedecinsController {
3.
4.     @Autowired
5.     private ApplicationModel application;
6.
7.     @Autowired
8.     private RdvMedecinsCorsController rdvMedecinsCorsController;
9.
10. ...
11.
12.     // liste des clients
13.     @RequestMapping(value = "/getAllClients", method = RequestMethod.GET)
14.     public Reponse getAllClients(HttpServletRequest response) {
15.         // entêtes CORS
16.         rdvMedecinsCorsController.getAllClients(response);
17.         // état de l'application
18.         if (messages != null) {
```

```

19.         return new Reponse(-1, messages);
20.     }
21.     // liste des clients
22.     try {
23.         return new Reponse(0, application.getAllClients());
24.     } catch (Exception e) {
25.         return new Reponse(1, Static.getErreursForException(e));
26.     }
27. }
28. ...

```

- ligne 8 : nous voulons réutiliser le code que nous avons placé dans le contrôleur [RdvMedecinsCorsController]. Aussi injectons-nous celui-ci ici ;
- ligne 14 : la méthode qui traite la demande [GET /getAllClients]. Nous faisons deux modifications :
 - ligne 14 : nous injectons l'objet [HttpServletRequest] dans les paramètres de la méthode,
 - ligne 16 : nous utilisons les méthodes de la classe [RdvMedecinsCorsController] pour mettre dans cet objet les entêtes CORS ;

3.7.7.6 Tests de l'application – 2

Nous lançons la nouvelle version du service web et redemandons la liste des clients. Nous obtenons la réponse suivante :

The screenshot shows two panels. On the left is a client application window titled "Rdvmedecins - v1". It has four input fields: "Temps d'attente : 0", "URL du service web : http://localhost:8080", "Login : admin", and "Mot de passe :". Below these is a blue button labeled "Liste des clients". On the right is a browser developer tools window titled "Liste des clients". It shows a dropdown menu with "Nothing selected" and a small number "2" in a yellow box above it. Below is a table in the "Network" tab of the developer tools. The table has columns: Name, Path, Method, and Status Text. It lists two entries: "getAllClients" (Method: OPTIONS, Status: 200 OK) and "getAllClients" (Method: GET, Status: 200 OK). A small number "3" is highlighted in a yellow box next to the second entry.

Name	Path	Method	Status Text
	getAllClients	OPTIONS	200 OK
	getAllClients	GET	200 OK

- en [1], on a bien une réponse mais elle est vide [2] ;
- en [3] : les échanges réseau se sont bien passés ;

Dans les logs console, la méthode [dao.getData] a affiché la réponse qu'elle a reçue :

```
[dao] getData[/getAllClients] success réponse : {"data": {"status": 0, "data": [{"id": 1, "version": 1, "titre": "Mr", "nom": "MARTIN", "prenom": "Jules"}, {"id": 2, "version": 1, "titre": "Mme", "nom": "GERMAN", "prenom": "Christine"}, {"id": 3, "version": 1, "titre": "Mr", "nom": "JACQUARD", "prenom": "Jules"}, {"id": 4, "version": 1, "titre": "Melle", "nom": "BISTROU", "prenom": "Brigitte"}]}, "status": 200, "config": {"method": "GET", "transformRequest": [null], "transformResponse": [null], "timeout": 1000, "url": "http://localhost:8080/getAllClients", "headers": {"Accept": "application/json, text/plain, */*", "Authorization": "Basic YWRtaW46YWRtaW4="}}, "statusText": "OK"}
```

Donc la méthode a bien reçu la liste des clients. Une fois le code vérifié, on en vient à suspecter l'instruction suivante qu'on ne maîtrise pas très bien :

```

1. // on style la liste déroulante
2. $('.selectpicker').selectpicker();

```

On met la ligne 2 en commentaires et on réessaie. On obtient alors la réponse suivante :

On a donc localisé le problème. C'est l'application de la méthode [selectpicker] à la liste déroulante qui pose problème. Lorsqu'on regarde le code source de la page erronée, on a la chose suivante :

```

<div class="row" style="margin-top: 20px" ng-show="clients.show">
  :before
  <div class="col-md-3">
    <h2 translate="msg_list_clients" class="ng-scope">Liste des clients</h2>
    <select data-style="btn-primary" class="selectpicker" select-enable style="display: none;">
      <!-- ngRepeat: client in clients.data -->
      <option ng-repeat="client in clients.data" value="1" class="ng-binding ng-scope">1
        Mr Jules MARTIN
      </option>
      <!-- end ngRepeat: client in clients.data -->
      <option ng-repeat="client in clients.data" value="2" class="ng-binding ng-scope">
        Mme Christine GERMAN
      </option>
      <!-- end ngRepeat: client in clients.data -->
      <option ng-repeat="client in clients.data" value="3" class="ng-binding ng-scope">
        Mr Jules JACQUARD
      </option>
      <!-- end ngRepeat: client in clients.data -->
      <option ng-repeat="client in clients.data" value="4" class="ng-binding ng-scope">
        Melle Brigitte BISTROU
      </option>
      <!-- end ngRepeat: client in clients.data -->
    </select>
    <div class="btn-group bootstrap-select">
      <button type="button" class="btn dropdown-toggle selectpicker btn-primary" data-toggle="dropdown" title="Nothing selected">
        <span class="filter-option pull-left">Nothing selected</span>
        &nbsp;
        <span class="caret"></span>
      </button>
      <div class="dropdown-menu open">
        <ul class="dropdown-menu inner selectpicker" role="menu"></ul>
      </div>
    </div>
  :after
</div>
<!-- la liste d'erreurs -->
<div class="alert alert-danger ng-binding ng-hide" ng-show="errors.show">...</div>
<!-- la liste d'erreurs -->
</div>

```

- on découvre qu'en [1], la liste déroulante est bien présente avec ses éléments mais qu'elle n'est pas affichée [style='display:none'] ;
- en [2], on voit le bouton [bootstrap select] affiché. Les éléments de la liste déroulante devraient apparaître dans la liste <ul role='menu'>. Ils n'y sont pas et on a donc une liste vide. Il semble que lorsque la méthode [selectpicker] a été appliquée à la liste déroulante, son contenu était vide à ce moment là ;

En parcourant la toile à la recherche d'une solution, on trouve celle-ci. On remplace le code :

```
1. // on style la liste déroulante
2. $('.selectpicker').selectpicker();
```

par le suivant :

```
1.           // on style la liste déroulante
2.           $timeout(function(){
3.               $('.selectpicker').selectpicker();
4.           });
```

Le style [bootstrap-select] est appliqué au travers d'une fonction [\$timeout]. Nous avons déjà rencontré cette fonction qui permet d'exécuter une fonction passé un certain délai. Ici, l'absence de délai vaut un délai nul. Les lignes précédentes mettent un événement dans la liste d'attente des événements du navigateur. Lorsque le traitement de l'événement en cours (clic sur le bouton [Liste des clients]) va être terminé, la vue V va être affichée. Puis aussitôt après, le navigateur va consulter sa liste d'événements. A cause de son délai nul, l'événement [\$timeout] va être en tête de liste et traité. Le style [bootstrap-select] est alors appliqué à une liste déroulante remplie. Voyons le résultat :

Rdvmedecins - v1

Temps d'attente :

URL du service web :

Login :

Mot de passe :

Liste des clients

2

Liste des clients

Mr Jules MARTIN

Mr Jules MARTIN

Mme Christine GERMAN

Mr Jules JACQUARD

Melle Brigitte BISTROU

Liste des clients

Mr Jules MARTIN

1

Si on regarde de nouveau le code source de la page affichée, on a la chose suivante :

```

<div class="btn-group bootstrap-select dropdown">
  <button type="button" class="btn dropdown-toggle selectpicker btn-primary" data-toggle="dropdown" title="Mr Jules MARTIN">...</button>
  <div class="dropdown-menu open" style="max-height: 424px; overflow: hidden; min-height: 92px;">
    <ul class="dropdown-menu inner selectpicker" role="menu" style="max-height: 412px; overflow-y: auto; min-height: 80px;">
      <li rel="0" class="selected">
        <a tabindex="0" class="ng-binding ng-scope" style="background-color: #f9f9f9; border-bottom: 1px solid #ccc; padding: 5px 10px; text-decoration: none; color: inherit;">
          <span class="text">Mr Jules MARTIN</span>
          <i class="glyphicon glyphicon-ok icon-ok check-mark"></i>
        </a>
      </li>
      <li rel="1">
        <a tabindex="0" class="ng-binding ng-scope" style="background-color: #f9f9f9; border-bottom: 1px solid #ccc; padding: 5px 10px; text-decoration: none; color: inherit;">
          <span class="text">Mme Christine GERMAN</span>
          <i class="glyphicon glyphicon-ok icon-ok check-mark"></i>
        </a>
      </li>
      <li rel="2">
        <a tabindex="0" class="ng-binding ng-scope" style="background-color: #f9f9f9; border-bottom: 1px solid #ccc; padding: 5px 10px; text-decoration: none; color: inherit;">
          <span class="text">Mr Jules JACQUARD</span>
          <i class="glyphicon glyphicon-ok icon-ok check-mark"></i>
        </a>
      </li>
      <li rel="3">
        <a tabindex="0" class="ng-binding ng-scope" style="background-color: #f9f9f9; border-bottom: 1px solid #ccc; padding: 5px 10px; text-decoration: none; color: inherit;">
          <span class="text">Melle Brigitte BISTRON</span>
          <i class="glyphicon glyphicon-ok icon-ok check-mark"></i>
        </a>
      </li>
    </ul>
  </div>
</div>
::after
</div>

```

1

Le bouton [bootstrap-select] qui précédemment était vide contient désormais la liste des clients.

3.7.7.7 Utilisation d'une directive

Nous avons rencontré dans le contrôleur C de la vue V, le code suivant :

```
// on style la liste déroulante
$('.selectpicker').selectpicker();
```

On manipule un objet du DOM. Nombre de développeurs Angular sont allergiques à la manipulation du DOM dans le code d'un contrôleur. Pour eux, celle-ci doit être faite dans une directive. Une directive Angular peut être vue comme une extension du langage HTML. Il est ainsi possible de créer de nouveaux éléments ou attributs HTML. Voyons un premier exemple :

Nous créons le fichier JS [selectEnable] suivant :

```

1. angular.module("rdvmedecins").directive('selectEnable', ['$timeout', function ($timeout) {
2.   return {
3.     link: function (scope, element, attrs) {
4.       $timeout(function () {
5.         var selectpicker = $('.selectpicker');
6.         selectpicker.selectpicker();
7.       });
8.     }
9.   };
10. }]);

```

- la directive suit la syntaxe du contrôleur à laquelle nous sommes désormais habitués :

```
angular.module("rdvmedecins").directive('selectEnable', ['$timeout', function ($timeout)
```

La directive appartient au module [rvmedecins]. C'est une fonction qui accepte deux paramètres :

- le premier est le nom de la directive [selectEnable] ;

- le second est un tableau ['obj1','obj2',..., fonction(obj1, obj2,...)] où les [obj] sont les objets à injecter dans la fonction. Ici le seul objet injecté est l'objet prédéfini [\$timeout] ;
- la fonction [directive] retourne un objet qui peut avoir divers attributs. Ici le seul attribut est l'attribut [link] (ligne 3). Sa valeur est ici une fonction admettant trois paramètres :
 - **scope** : le modèle de la vue dans laquelle est utilisée la directive ;
 - **element** : l'élément de la vue, objet de la directive ;
 - **attrs** : les attributs de cet élément ;

Prenons un exemple. La directive [selectEnable] pourrait être utilisée dans le contexte suivant :

```
<div select-enable="data"></div>
```

Ci-dessus, l'attribut [select-enable] applique la directive [selectEnable] à l'élément HTML <div>. Une directive [doSomething] peut être appliquée à n'importe quel élément HTML en lui ajoutant l'attribut [do-something]. On fera attention au changement d'écriture entre le nom de la directive et l'attribut qui lui est associé. On passe d'une écriture [camelCase] à une écriture [camel-case].

La directive [selectEnable] pourrait également utilisée de la façon suivante :

```
<select-enable attr1='val1' attr2='val2' ...>...</select-enable>
```

Ici la directive [doSomething] est appliquée sous la forme d'une balise HTML <do-something>.

Revenons à l'écriture

```
<div select-enable="data"></div>
```

et aux trois paramètres de la fonction [link] de la directive, [scope, element, attrs] :

- **scope** : est le modèle de la vue dans laquelle se trouve la <div> ;
- **element** : est la <div> elle-même ;
- **attrs** : est le tableau des attributs de la <div>. Ceux-ci peuvent être utilisés pour transmettre de l'information à la directive. Ci-dessus, on écrira **attrs['selectEnable']** pour avoir l'information [data]. On notera bien le changement d'écriture [selectEnable] pour désigner l'attribut [select-enable] ;

Revenons au code de la directive :

```
11. angular.module("rdvmedecins").directive('selectEnable', ['$timeout', function ($timeout) {
12.   return {
13.     link: function (scope, element, attrs) {
14.       $('.selectpicker').selectpicker();
15.     }
16.   };
17. }]);
```

- ligne 14 : on retrouve le code que nous avions placé auparavant dans le contrôleur. Celui-ci est exécuté lors de la rencontre de la directive [select-enable] (sous forme d'élément ou d'attribut) lors de l'affichage de la vue V.

Pour mettre en oeuvre cette directive, nous copions le fichier [app-17.html] dans [app-17B.html] et le modifions de la façon suivante :

```
1.   <select data-style="btn-primary" class="selectpicker" select-enable="">
2.     <option ng-repeat="client in clients.data" value="{{client.id}}>
3.       {{client.titre}} {{client.prenom}} {{client.nom}}
4.     </option>
5.   </select>
```

- ligne 1 : on applique la directive [selectEnable] à l'élément HTML [select]. Comme il n'y a pas d'informations à passer à la directive, nous écrivons simplement [select-enable=""] ;

Nous modifions également le contrôleur en dupliquant le fichier JS [rdvmedecins-05.js] dans [rdvmedecins-05B.js] et nous référençons le nouveau fichier JS dans le fichier [app-17B.html] et le fichier [selectEnable.js] de directive. Il ne faut pas oublier ce dernier point. Si le fichier de la directive est absent, l'attribut [select-enable=""] ne sera pas géré mais Angular ne signalera aucune erreur.

```

1. <script type="text/javascript" src="rdvmedecins-05B.js"></script>
2. <script type="text/javascript" src="selectEnable.js"></script>

```

Dans le fichier JS [rdvmedecins-05B.js], nous supprimons du contrôleur, les lignes suivantes :

```

1.           // on style la liste déroulante
2.           $timeout(function(){
3.               $('.selectpicker').selectpicker();
4.           });

```

cette opération étant désormais faite par la directive.

3.7.7.8 Tests de l'application – 3

Lorsqu'on teste la nouvelle application [app-17B.html], on obtient le résultat suivant :

- en [1], on obtient une liste vide.

Les logs console affichent la chose suivante :

```

1. [dao] init
2. directive selectEnable
3. [dao] getData[/getAllClients] success réponse : {"data":{"status":0,"data": [{"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"}, {"id":2,"version":1,"titre":"Mme","nom":"GERMAN","prenom":"Christine"}, {"id":3,"version":1,"titre":"Mr","nom":"JACQUARD","prenom":"Jules"}, {"id":4,"version":1,"titre":"Melle","nom":"BISTRON","prenom":"Brigitte"}]},{"status":200,"config":{"method":"GET","transformRequest":[null],"transformResponse": [null],"timeout":1000,"url":"http://localhost:8080/getAllClients","headers":{"Accept":"application/json, text/plain, */*","Authorization":"Basic YWRtaW46YWRtaW4="}},"statusText":"OK"}

```

- ligne 1 : initialisation du service [dao] ;
- ligne 2 : à l'affichage initial de la vue V, la directive [selectEnable] est exécutée ;
- ligne 3 : cette ligne apparaît lorsque l'utilisateur clique sur le bouton [Liste des clients]. On constate alors que la directive [selectEnable] n'est pas exécutée une seconde fois. Au final, elle a été exécutée lorsque la liste des clients était vide et on a donc une liste déroulante vide ;

Dit autrement, l'opération :

```
1. $('.selectpicker').selectpicker();
```

ne s'est pas déroulée au bon moment. On peut essayer de résoudre le problème de diverses façons. Au bout de nombreux tests infructueux, on se rend compte que l'opération ci-dessus **ne doit se dérouler qu'une fois et uniquement lorsque la liste déroulante a été remplie**. Pour obtenir ce résultat, on réécrit la balise <select> de la façon suivante :

```
1.      <select data-style="btn-primary" class="selectpicker" select-enable="" ng-if="clients.data">
2.          <option ng-repeat="client in clients.data" value="{{client.id}}>
3.              {{client.titre}} {{client.prenom}} {{client.nom}}
4.          </option>
5.      </select>
```

Ligne 1, la balise <select> n'est générée que si [clients.data] existe. Ce n'est pas le cas lors de l'affichage initial de la vue V. La balise <select> ne sera donc pas générée et la directive [selectEnable] pas évaluée. Lorsque l'utilisateur va cliquer sur le bouton [Liste des clients], [clients.data] aura une nouvelle valeur dans le modèle M. Parce que le modèle M a changé, la balise <select> va être réévaluée et ici générée. La directive [selectEnable] va donc être évaluée également. Lorsqu'elle est évaluée, les lignes 2-4 de la balise <select> n'ont pas encore été évaluées. On a donc une liste de clients vide. Si on écrit la directive [selectEnable] de la façon suivante :

```
1. angular.module("rdvmedecins").directive('selectEnable', ['$timeout', 'utils', function ($timeout, utils) {
2.     return {
3.         link: function (scope, element, attrs) {
4.             utils.debug("directive selectEnable");
5.             $('.selectpicker').selectpicker();
6.         }
7.     }
8. }]);
```

la ligne 5 va être exécutée avec une liste vide et on aura alors une liste déroulante vide à l'affichage. Il faut alors écrire :

```
1. angular.module("rdvmedecins").directive('selectEnable', ['$timeout', 'utils', function ($timeout, utils) {
2.     return {
3.         link: function (scope, element, attrs) {
4.             utils.debug("directive selectEnable");
5.             $timeout(function () {
6.                 $('.selectpicker').selectpicker();
7.             })
8.         }
9.     }
10. }]);
```

pour avoir le résultat attendu. A cause du [\$timeout] de la ligne 5, la ligne 6 ne sera exécutée qu'après évaluation complète de la vue V, donc à un moment où la balise <select> aura tous ses éléments.

3.7.8 Exemple 8 : l'agenda d'un médecin

Nous présentons maintenant une application qui affiche l'agenda d'un médecin.

3.7.8.1 La vue V de l'application

Nous présenterons le formulaire suivant :

Rdvmedecins - v1

The screenshot shows a web application interface. On the left, a dropdown menu displays "Mme Marie PELISSIER". Below it, a button labeled "Agenda" is highlighted with a yellow box [1]. In the center, a title "Liste des médecins" is displayed above a date picker for June 2014. The date "25" is selected and highlighted with a yellow box [2]. The day "25" is also highlighted with a yellow box [3] in the calendar grid. The "Agenda" button is also highlighted with a yellow box [4].

- en [1], on demande l'agenda de Mme PELISSIER [2], le 25 juin 2014 [3] ;

On obtient le résultat [4] suivant :

Créneau horaire	Client	Action
08h00:08h20		Réserver
08h20:08h40	Mr Jules MARTIN	Supprimer
08h40:09h00		Réserver
09h00:09h20		Réserver
09h20:09h40	4	Réserver
09h40:10h00		Réserver

Nous allons étudier les deux vues séparément.

3.7.8.2 Le formulaire

Nous dupliquons le fichier [app-17.html] dans [app-18.html] puis nous modifions le code de la façon suivante :

```
1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <!-- le message d'attente -->
5.   <div class="alert alert-warning" ng-show="waiting.visible">
6.     ...
7.   </div>
8.
9.   <!-- la demande -->
10.  <div class="alert alert-info" ng-hide="waiting.visible">
11.    <div class="row" style="margin-bottom: 20px">
12.      <div class="col-md-3">
13.        <h2 translate="{{medecins.title}}></h2>
14.        <select data-style="btn-primary" class="selectpicker">
15.          <option ng-repeat="medecin in medecins.data" value="{{medecin.id}}>
16.            {{medecin.titre}} {{medecin.prenom}} {{medecin.nom}}</option>
```

```

17.          </option>
18.      </select>
19.  </div>
20.  <div class="col-md-3">
21.      <h2 translate="{{calendar.title}}></h2>
22.      <div style="display:inline-block; min-height:290px;">
23.          <datepicker ng-model="calendar.jour" min-date="calendar.minDate" show-weeks="true"
24.                      class="well well-sm"></datepicker>
25.      </div>
26.  </div>
27. </div>
28. <button class="btn btn-primary" ng-click="execute()">{{agenda.title|translate}}</button>
29. </div>
30.
31. <!-- la liste d'erreurs -->
32. <div class="alert alert-danger" ng-show="errors.show">
33. ...
34. </div>
35.
36. <!-- l'agenda -->
37. <div id="agenda" ng-show="agenda.show">
38. ...
39. </div>
40. </div>
41. ...
42. <script type="text/javascript" src="rdvmedecins-06.js"></script>

```

- lignes 5-7 : le message d'attente ne change pas ;
- lignes 12-19 : la liste des médecins de type [bootstrap select] ;
- lignes 20-26 : le calendrier de [ui-bootstrap] que nous avons déjà présenté. On notera que le jour sélectionné est placé dans le modèle [calendar.jour] (attribut ng-model) ;
- ligne 28 : le bouton qui demande l'agenda ;
- lignes 32-34 : la liste des erreurs ne change pas ;
- lignes 37-39 : l'agenda que nous présenterons ultérieurement ;
- ligne 42 : le code JS est transféré dans le fichier [rdvmedecins-06.js] par recopie du fichier [rdvmedecins-05.js] ;

3.7.8.3 Le contrôleur C

Le code JS de l'application devient le suivant :

```

5   // -----
6   // ----- module Angular
7   angular.module("rdvmedecins", [...]);
8
9   // configuration initiale
10  angular.module("rdvmedecins")
11  + .config(['$translateProvider', function ($translateProvider) {...}]);
12
13  // services
14  angular.module("rdvmedecins")
15  + .factory('config', function () {...})
16
17
18  angular.module("rdvmedecins")
19  + .factory('utils', ['config', '$timeout', '$q', function (config, $timeout, $q) {...}]);
20
21  angular.module("rdvmedecins")
22  + .factory('dao', [...]);
23
24
25  // contrôleur
26  angular.module("rdvmedecins")
27  + .controller('rdvMedecinsCtrl', [...])
28
29

```

Seuls le service [utils] et le contrôleur [rdvMedecinsCtrl] vont être impactés par les modifications.

Le contrôleur [rdvMedecinsCtrl] devient le suivant :

```
1. // contrôleur
```

```

2. angular.module("rdvmedecins")
3.   .controller('rdvMedecinsCtrl', ['$scope', 'utils', 'config', 'dao', '$translate', '$timeout', '$filter',
4.   '$locale',
5.     function ($scope, utils, config, dao, $translate, $timeout, $filter, $locale) {
6.       // ----- initialisation modèle
7.       // modèle
8.       $scope.waiting = {text: config.msgWaiting, visible: false, cancel: cancel, time: 3000};
9.       $scope.server = {url: 'http://localhost:8080', login: 'admin', password: 'admin'};
10.      $scope.errors = {show: false, model: {}};
11.      $scope.medecins = {
12.        data: [
13.          {id: 1, version: 1, titre: "Mme", nom: "PELISSIER", prenom: "Marie"},
14.          {id: 2, version: 1, titre: "Mr", nom: "BROMARD", prenom: "Jacques"},
15.          {id: 3, version: 1, titre: "Mr", nom: "JANDOT", prenom: "Philippe"},
16.          {id: 4, version: 1, titre: "Melle", nom: "JACQUEMOT", prenom: "Justine"}
17.        ],
18.        title: config.listMedecins};
19.      $scope.agenda = {title: config.getAgendaTitle, data: undefined, show: false};
20.      $scope.calendar = {title: config.getCalendarTitle, minDate: new Date(), jour: new Date()};
21.      // on style la liste déroulante
22.      $timeout(function () {
23.        $('.selectpicker').selectpicker();
24.      });
25.      // locale française pour le calendrier
26.      angular.copy(config.locales['fr'], $locale);
27.    ...
28.  })
29. ;

```

- ligne 7 : on fixe un temps d'attente de 3 secondes avant de faire l'appel HTTP ;
- ligne 8 : on fixe en dur les éléments nécessaires à la connexion HTTP ;
- lignes 10-17 : la liste des médecins est définie en dur ;
- ligne 18 : le modèle [agenda] configure l'affichage de l'agenda dans la vue ;
- ligne 19 : le modèle [calendar] configure l'affichage du calendrier dans la vue. On fixe une date minimale [minDate] à aujourd'hui et la date du jour également à aujourd'hui ;
- lignes 21-23 : la liste déroulante est stylée avec la méthode vue précédemment ;
- ligne 25 : on met la locale de l'application à 'fr'. Par défaut, elle est à 'en' ;

La méthode exécutée lors de la demande de l'agenda est la suivante :

```

1. // exécution action
2.   $scope.execute = function () {
3.     // les infos du formulaire
4.     var idMedecin = $('.selectpicker').selectpicker('val');
5.
6.     // vérification
7.     utils.debug("[homeCtrl] idMedecin", idMedecin);
8.     utils.debug("[homeCtrl] jour", $scope.calendar.jour);
9.
10.    // on met le jour au format yyyy-MM-dd
11.    var formattedJour = $filter('date')($scope.calendar.jour, 'yyyy-MM-dd');
12.    // mise à jour de la vue
13.    $scope.waiting.visible = true;
14.    $scope.errors.show = false;
15.    $scope.agenda.show = false;
16.  ...
17. };

```

- ligne 4 : on récupère l'attribut [value] du médecin sélectionné. On utilise ici de nouveau la méthode [selectpicker] qui provient du fichier [\[bootstrap-select.min.js\]](#). Il faut se souvenir de la forme des options de la liste déroulante :

```

<option ng-repeat="medecin in medecins.data" value="{{medecin.id}}">
  {{medecin.titre}} {{medecin.prenom}} {{medecin.nom}}

```

La valeur (attribut *value*) de l'option est donc l'identifiant [id] du médecin.

- ligne 11 : on met le jour choisi par l'utilisateur au format [aaaa-mm-jj] qui est le format de date attendu par le serveur web ;
- lignes 13-15 : lorsque la méthode [execute] sera terminée, le bandeau d'attente sera affiché et tout le reste caché ;

Le code se poursuit de la façon suivante :

```
1. // attente simulée
2.     var task = utils.waitForSomeTime($scope.waiting.time);
3.     // on demande l'agenda du médecin
4.     var promise = task.promise.then(function () {
5.         // le chemin de l'URL de service
6.         var path = config.urlSrvAgenda + "/" + idMedecin + "/" + formattedJour;
7.         // on demande l'agenda
8.         task = dao.getData($scope.server.url, $scope.server.login, $scope.server.password,
    path);
9.         // on retourne la promesse d'achèvement de la tâche
10.        return task.promise;
11.    });
12.    // on analyse le résultat de l'appel au service [dao]
13.    promise.then(function (result) {
14.        // fin d'attente
15.        $scope.waiting.visible = false;
16.        // erreur ?
17.        if (result.err == 0) {
18.            // on prépare le modèle de l'agenda
19.            $scope.agenda.data = result.data;
20.            $scope.agenda.show = true;
21.            // mise en forme de l'affichage des horaires
22.            angular.forEach($scope.agenda.data.creneauxMedecin, function (creneauMedecin) {
23.                creneauMedecin.creneau.text = utils.getTextForCreneau(creneauMedecin.creneau);
24.            });
25.            // on crée un evt pour styler la table après l'affichage de la vue
26.            $timeout(function () {
27.                $("#creneaux").footable();
28.            });
29.        } else {
30.            // il y a eu des erreurs pour obtenir l'agenda
31.            $scope.errors = {
32.                title: config.getAgendaErrors,
33.                messages: utils.getErrors(result),
34.                show: true
35.            };
36.        }

```

- ligne 2 : la tâche asynchrone d'attente de 3 secondes ;
- lignes 5-10 : le code qui sera exécuté lorsque cette attente sera terminée ;
- ligne 6 : on construit l'URL interrogée [/getAgendaMedecinJour/1/2014-06-25] ;
- ligne 8 : l'URL est interrogée. Une tâche asynchrone démarre ;
- ligne 10 : on rend la promesse de cette tâche asynchrone ;
- lignes 14-38 : le code qui sera exécuté lorsque l'appel HTTP aura renvoyé sa réponse ;
- ligne 13 : [result] est la réponse envoyée par la méthode [dao.getData]. Il faut ici se souvenir de la forme de la réponse du serveur web :

```

{
  "status": 0,
  "data": {
    "medecin": {
      "prenom": "Marie",
      "id": 1,
      "nom": "PELISSIER",
      "titre": "Mme"
    },
    "creneauxMedecin": [
      {"rv": null, "creneau": {"id": 1, "mDebut": 0, "mFin": 20, "hFin": 8, "hDebut": 8}}, {"rv": null, "creneau": {"id": 2, "mDebut": 1, "mFin": 20, "hFin": 40, "hDebut": 8}}, {"rv": null, "creneau": {"id": 3, "mDebut": 40, "mFin": 0, "hFin": 9, "hDebut": 8}}, {"rv": null, "creneau": {"id": 4, "mDebut": 0, "mFin": 20, "hFin": 9, "hDebut": 9}}, {"rv": {"id": 115, "client": {"prenom": "Jules", "id": 1, "nom": "MARTIN", "titre": "Mr"}}, "creneau": {"id": 5, "mDebut": 20, "mFin": 40, "hFin": 9, "hDebut": 9}}, {"creneau": {"id": 5, "mDebut": 20, "mFin": 40, "hFin": 9, "hDebut": 9}}, {"rv": null, "creneau": {"id": 7, "mDebut": 0, "mFin": 20, "hFin": 10, "hDebut": 10}}, {"rv": null, "creneau": {"id": 8, "mDebut": 20, "mFin": 40, "hFin": 10, "hDebut": 10}}, {"rv": null, "creneau": {"id": 9, "mDebut": 40, "mFin": 0, "hFin": 11, "hDebut": 10}}, {"rv": null, "creneau": {"id": 10, "mDebut": 0, "mFin": 20, "hFin": 11, "hDebut": 11}}, {"rv": null, "creneau": {"id": 11, "mDebut": 20, "mFin": 40, "hFin": 11, "hDebut": 11}}, {"rv": null, "creneau": {"id": 12, "mDebut": 40, "mFin": 0, "hFin": 12, "hDebut": 11}}, {"rv": null, "creneau": {"id": 13, "mDebut": 0, "mFin": 20, "hFin": 14, "hDebut": 14}}, {"rv": null, "creneau": {"id": 14, "mDebut": 20, "mFin": 40, "hFin": 14, "hDebut": 14}}, {"rv": null, "creneau": {"id": 15, "mDebut": 40, "mFin": 0, "hFin": 15, "hDebut": 14}}, {"rv": null, "creneau": {"id": 16, "mDebut": 0, "mFin": 20, "hFin": 15, "hDebut": 15}}, {"rv": null, "creneau": {"id": 17, "mDebut": 20, "mFin": 40, "hFin": 15, "hDebut": 15}}, {"rv": null, "creneau": {"id": 18, "mDebut": 40, "mFin": 0, "hFin": 16, "hDebut": 15}}, {"rv": null, "creneau": {"id": 19, "mDebut": 0, "mFin": 20, "hFin": 16, "hDebut": 16}}, {"rv": null, "creneau": {"id": 20, "mDebut": 20, "mFin": 40, "hFin": 16, "hDebut": 16}}, {"rv": null, "creneau": {"id": 21, "mDebut": 40, "mFin": 0, "hFin": 17, "hDebut": 16}}, {"rv": null, "creneau": {"id": 22, "mDebut": 0, "mFin": 20, "hFin": 17, "hDebut": 17}}, {"rv": null, "creneau": {"id": 23, "mDebut": 20, "mFin": 40, "hFin": 17, "hDebut": 17}}, {"rv": null, "creneau": {"id": 24, "mDebut": 40, "mFin": 0, "hFin": 18, "hDebut": 17}}], "jour": "2014-06-12"
}

```

Le paramètre [result.data] de la ligne 19 est l'attribut [data] [1] ci-dessus. Cet attribut contient à son tour l'attribut [creneauxMedecin] [2] ci-dessus. Celui-ci est un tableau de créneaux avec pour chacun d'eux les deux informations :

- [rv] : la forme JSON d'un rendez-vous ou [null] s'il n'y a pas de rendez-vous pris sur ce créneau ;
- [hDeb, mDeb, hFin, mFin] : les informations horaires du créneau ;

Revenons au code du contrôleur :

- ligne 15 : l'attente est terminée ;
- ligne 19 : on renseigne le modèle [\$scope.agenda] qui contrôle l'affichage de l'agenda ;
- ligne 20 : l'agenda est rendu visible ;
- lignes 22-24 : on passe en revue chacun des éléments C du tableau [creneauxMedecin] dont nous venons de parler ;
- ligne 23 : chaque élément C a un attribut [creneau] qui est le créneau horaire. Celui-ci est enrichi d'un attribut [text] qui sera la représentation texte du créneau horaire sous la forme [10h20:10h40] ;
- lignes 26-28 : on rend 'responsive' la table HTML utilisée pour afficher les créneaux de l'agenda. Nous avons vu cette notion au paragraphe 3.6.7, page 152 ;

Créneau horaire	Client
+ 08h00:08h20	
+ 08h20:08h40	Mr Jules MARTIN
+ 08h40:09h00	
+ 09h00:09h20	

- ligne 27 : pour rendre la table 'responsive', il faut lui appliquer la méthode [footable]. On retrouve ici la même difficulté que celle rencontrée pour le composant [bootstrap-select]. Si on écrit simplement la ligne 17, on constate que la table n'est pas 'responsive'. On résoud ce problème de la même façon avec la fonction [\$timeout] (ligne 26) ;
- lignes 31-34 : le cas où l'appel HTTP a échoué. On affiche alors les messages d'erreur ;

3.7.8.4 Affichage de l'agenda

Nous revenons maintenant au code de l'agenda dans le fichier [app-18.html]. C'est le suivant :

```

1. <!-- l'agenda -->
2. <div id="agenda" ng-show="agenda.show">
3.   <!-- cas du médecin sans créneaux de consultation -->
4.   <h4 class="alert alert-danger" ng-if="agenda.data.creneauxMedecin.Length==0"
5.     translate="agenda_medecinsanscreneaux"></h4>
6.   <!-- agenda du médecin -->
7.   <div class="row tab-content alert alert-warning" ng-if="agenda.data.creneauxMedecin.Length!=0">
8.     <div class="tab-pane active col-md-6">
9.       <table creneaux-table id="creneaux" class="table">
10.         <thead>
11.           <tr>
12.             <th data-toggle="true">
13.               <span translate="agenda_creneauhoraire"></span>
14.             </th>
15.             <th>
16.               <span translate="agenda_client">Client</span>
17.             </th>
18.             <th data-hide="phone">
19.               <span translate="agenda_action">Action</span>
20.             </th>
21.           </tr>
22.         </thead>
23.         <tbody>
24.           <tr ng-repeat="creneauMedecin in agenda.data.creneauxMedecin">
25.             <td>
26.               <span
27.                 ng-class="! creneauMedecin.rv ? 'status-metro status-active' : 'status-metro status-
suspended'">
28.                 {{creneauMedecin.creneau.text}}
29.               </span>
30.             </td>
31.             <td>
32.               <span>{{creneauMedecin.rv.client.titre}} {{creneauMedecin.rv.client.prenom}}
{{creneauMedecin.rv.client.nom}}</span>
33.             </td>
34.             <td>
35.               <a href="" ng-if="!creneauMedecin.rv" translate="agenda_reserver" class="status-metro
status-active">
36.                 </a>
37.               <a href="" ng-if="creneauMedecin.rv" translate="agenda_supprimer" class="status-metro
status-suspended">
38.                 </a>
39.               </td>
40.             </tr>
41.           </tbody>
42.         </table>
43.       </div>
44.     </div>
45. </div>
```

- lignes 4-5 : on se rappelle que [agenda.data] est l'agenda, que [agenda.data.creneauxMedecin] est un tableau d'objets de type [CreneauMedecin]. Chaque élément de ce dernier type a un attribut [CreneauMedecin].creneau qui est un créneau horaire. Chaque créneau horaire a deux éléments qui nous intéressent :
 - [creneauMedecin.creneau.rv] qui est l'éventuel RV (rv!=null) pris sur le créneau ;
 - [creneauMedecin.creneau.text] qui est le texte [début:fin] du créneau horaire ;
- ligne 4 : affiche un message spécial si le médecin n'a pas de créneaux horaires. C'est improbable mais il se trouve que notre base de données est incomplète et ce cas existe. La génération HTML ou non du message est contrôlée par la directive [ng-if] ;

Rdvmedecins - v1

The screenshot shows a web application interface for managing medical appointments. On the left, there's a sidebar titled "Liste des médecins" with a dropdown menu showing "Melle Justine JACQUEMOT". In the center, there's a "Jour" (Day) section featuring a calendar for June 2014. The calendar highlights the 25th of June in blue. Below the calendar, a button labeled "Agenda" is visible. A red-shaded box at the bottom contains the text "Ce médecin n'a pas encore de créneaux de consultation" (This doctor has not yet scheduled any appointment slots).

La directive [ng-if] est différente des directives [ng-show, ng-hide]. Ces dernières se contentent de cacher une zone présente dans le document. Si [ng-if='false'], alors la zone est enlevée du document. On l'a utilisée ici pour illustration ;

- ligne 9 : l'attribut [id='creneaux'] est important. C'est lui qui est utilisé dans l'instruction :

```
$("#creneaux").footable();
```

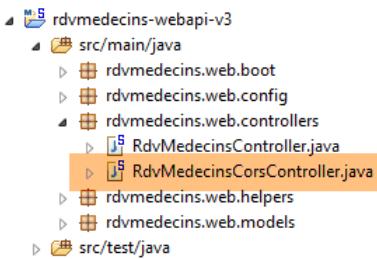
- lignes 10-22 : affichent les entêtes du tableau [1] ;
- lignes 23-45 : affichent le contenu du tableau [2] ;

Créneau horaire	1	Client	Action
08h00:08h20	1		Réserver
08h20:08h40	3	Mr Jules MARTIN	Supprimer
08h40:09h00	2		Réserver

- ligne 24 : on parcourt le tableau [`agenda.data.creneauxMedecin`] ;
- lignes 26-29 : on écrit le texte [3]. On utilise la directive [ng-class] qui va générer l'attribut [class] de l'élément. Ici, si on a [`creneauMedecin.rv==null`], cela veut dire que le créneau est libre et on met un fond vert au texte. Sinon, on met un fond rouge ;
- ligne 32 : on écrit le nom du client pour qui a été pris le RV [4]. Si [`rv==null`], ces informations n'existent pas mais Angular gère correctement ce cas et ne déclare pas d'erreur ;
- lignes 34-39 : affichent l'un des deux boutons [Réserver] ou [Supprimer]. C'est l'existence ou non d'un rendez-vous qui fait que l'on choisit l'un ou l'autre des boutons ;

3.7.8.5 Modification du serveur web

Comme pour les exemples précédents, le serveur web doit être modifié pour que l'URL [/getAgendaMedecinJour] envoie les entêtes CORS :



Dans la classe [RdvMedecinsCorsController] on ajoute une nouvelle méthode :

```

1.    // agenda du médecin
2.    @RequestMapping(value = "/getAgendaMedecinJour/{idMedecin}/{jour}", method =
   RequestMethod.OPTIONS)
3.    public void getAgendaMedecinJour(HttpServletRequest response) {
4.        sendOptions(response);
5.    }

```

Cette méthode va envoyer les entêtes CORS pour la requête HTTP [OPTIONS]. On doit faire la même chose pour la requête HTTP [GET] dans la classe [RdvMedecinsController] :

```

1. @RequestMapping(value = "/getAgendaMedecinJour/{idMedecin}/{jour}", method = RequestMethod.GET)
2.    public Reponse getAgendaMedecinJour(@PathVariable("idMedecin") long idMedecin,
   @PathVariable("jour") String jour, HttpServletRequest response) {
3.        // entêtes CORS
4.        rdvMedecinsCorsController.getAgendaMedecinJour(response);
5.    ...
6. }

```

3.7.8.6 Utilisation de directives

Comme il a été fait précédemment, nous allons déporter la manipulation du DOM dans des directives. Nous avons deux manipulations de DOM :

- lors de l'affichage initial de la vue :

```

1.    // on style la liste déroulante
2.    $timeout(function () {
3.        $('.selectpicker').selectpicker();
4.    });

```

- lors de l'affichage de l'agenda :

```

1.          // on crée un evt pour styler la table après l'affichage de la vue
2.          $timeout(function () {
3.              $("#creneaux").footable();
4.          });

```

Pour le 1er cas, nous allons utiliser la directive [selectEnable] déjà présentée. Pour le second cas, nous créons la directive [footable] dans le fichier JS [footable.js] suivant :

```

1. angular.module("rdvmedecins").directive('footable', ['$timeout', 'utils', function ($timeout, utils) {
2.     return {
3.         link: function (scope, element, attrs) {
4.             utils.debug("directive footable");
5.             $timeout(function () {
6.                 $("#creneaux").footable();
7.             })
8.         }
9.     }
10. }]);

```

On utilise donc la même technique que pour la directive [selectEnable].

Le code HTML [app-18.html] est dupliqué dans [app-18B.html]. Puis on le fait évoluer de la façon suivante :

```
1.      <select data-style="btn-primary" class="selectpicker" select-enable="">
2.          <option ng-repeat="medecin in medecins.data" value="{{medecin.id}}">
3.              {{medecin.titre}} {{medecin.prenom}} {{medecin.nom}}
4.          </option>
5.      </select>
```

- ligne 1 : on applique la directive [selectEnable] (via l'attribut [select-enable]) à la balise <select> des médecins ;

```
1.      <div class="row tab-content alert alert-warning" ng-if="agenda.data.creneauxMedecin.Length!=0">
2.          <div class="tab-pane active col-md-6">
3.              <table id="creneaux" class="table" footable="">
4.                  <thead>
5.                      <tr>
```

- ligne 3 : on applique la directive [footable] (via l'attribut [footable]) à la table HTML de l'agenda ;

```
1.  <script type="text/javascript" src="rdvmedecins-06B.js"></script>
2.  <!-- directives -->
3.  <script type="text/javascript" src="selectEnable.js"></script>
4.  <script type="text/javascript" src="footable.js"></script>
```

- lignes 3-4 : on référence les fichiers JS des deux directives ;
- ligne 1 : le code JS de [app-18B.html] est le code JS de [app-18.html] dupliqué dans le fichier [[rdvmedecins-06B.js](#)] ;

Le fichier [[rdvmedecins-06B.js](#)] est identique au fichier [[rdvmedecins-06.js](#)] à deux détails près. Les lignes manipulant le DOM disparaissent :

```
1.      // on style la liste déroulante
2.      $timeout(function () {
3.          $('.selectpicker').selectpicker();
4.      });
```

```
1.          // on crée un evt pour styler la table après l'affichage de la vue
2.          $timeout(function () {
3.              $("#creneaux").footable();
4.          });
```

Ceci fait, l'exécution de l'application [app-18B.html] donne les mêmes résultats que celle de [app-18.html].

3.7.9 Exemple 9 : créer et annuler des réservations

Nous présentons maintenant une application qui permet de créer et d'annuler des réservations.

3.7.9.1 La vue V de l'application

Nous présenterons le formulaire suivant :

Rdvmedecins - v1

Agenda de Mme Marie PELISSIER le jeudi 26 juin 2014

Créneau horaire	Client	Action
08h00:08h20	Mme Christine GERMAN	Supprimer 2
08h20:08h40		Réserver
08h40:09h00		Réserver
09h00:09h20	Mr Jules JACQUARD	Supprimer
09h20:09h40		Réserver 1
09h40:10h00		Réserver
10h00:10h20		Réserver

- en [1], on pourra réserver. La réservation qui sera faite le sera pour un client aléatoire ;
- en [2], on pourra supprimer les réservations que nous aurons faites ;

Nous dupliquons le fichier [app-18.html] dans [app-19.html] puis nous modifions le code de la façon suivante :

```
1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <!-- le message d'attente -->
5.   <div class="alert alert-warning" ng-show="waiting.visible">
6.   ...
7.   </div>
8.
9.   <!-- la liste d'erreurs -->
10.  <div class="alert alert-danger" ng-show="errors.show">
11.  ...
12.  </div>
13.
14.  <!-- l'agenda -->
15.  <div id="agenda" ng-show="agenda.show">
16.  ...
17.    <!-- agenda du médecin -->
18.    <div class="row tab-content alert alert-warning" ng-if="agenda.data.creneauxMedecin.Length!=0">
19.      <div class="tab-pane active col-md-6">
20.        <table id="creneaux" class="table footable="">
21.        ...
22.          <tbody>
23.            <tr ng-repeat="creneauMedecin in agenda.data.creneauxMedecin">
24.            ...
25.              <td>
26.                <a href="" ng-if="!creneauMedecin.rv" translate="agenda_reserver" class="status-metro status-active" ng-click="reserver(creneauMedecin.creneau.id)">
27.                  </a>
28.                <a href="" ng-if="creneauMedecin.rv" translate="agenda_supprimer" class="status-metro status-suspended" ng-click="supprimer(creneauMedecin.rv.id)">
29.                  </a>
30.                </td>
31.              </tr>
32.            </tbody>
33.          </table>
34.        </div>
35.      </div>
36.    </div>
37.  </div>
38.  ....
```

```

39. <script type="text/javascript" src="rdvmedecins-07.js"></script>
40. <script type="text/javascript" src="footable.js"></script>

```

- lignes 5-7 : le message d'attente est celui de la version précédente ;
- lignes 10-12 : le message d'erreurs est celui de la version précédente ;
- lignes 15-36 : l'agenda est celui de la version précédente à deux détails près :
 - ligne 26 : le clic sur le bouton [réserver] (attribut ng-click) est géré par la méthode [reserver] du modèle M de la vue V. On lui passe le n° du créneau horaire de réservation ;
 - ligne 26 : le clic sur le bouton [supprimer] est géré par la méthode [reserver] du modèle M de la vue V. On lui passe le n° du rendez-vous à supprimer ;
- ligne 39 : le code JS qui gère l'application est dans le fichier [rdvmedecins-07.js] ;
- ligne 40 : le code JS de la directive [footable] appliquée ligne 20 ;

3.7.9.2 Le contrôleur C

Le code JS de [rdvmedecins-07.js] est d'abord obtenu par recopie du fichier [rdvmedecins-06.js]. Puis il est modifié. On a toujours les grands blocs de code habituels. Les modifications se font essentiellement dans le contrôleur :

```

5   // ----- module Angular
6   angular.module("rdvmedecins", [ 'pascalprecht.translate', 'base64', 'ngLocale', 'ui.bootstrap' ]);
7
8   // configuration initiale
9   angular.module("rdvmedecins")
10  + .config(['$translateProvider', function ($translateProvider) {...}]);
11
12  // services
13  +angular.module("rdvmedecins").factory('config', function () {...});
14
15
16  angular.module("rdvmedecins")
17  + .factory('utils', ['$config', '$timeout', '$q', function ($config, $timeout, $q) {...}]);
18
19
20  angular.module("rdvmedecins").factory(
21    'dao', [...]);
22
23
24  // contrôleur
25  angular.module("rdvmedecins")
26  + .controller('rdvMedecinsCtrl', [...]);
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

```

Nous allons décrire le contrôleur C de la vue V en plusieurs étapes.

3.7.9.3 Initialisation du contrôleur C

Le code d'initialisation du contrôleur est le suivant :

```

1. angular.module("rdvmedecins")
2.   .controller('rdvMedecinsCtrl', ['$scope', 'utils', 'config', 'dao', '$translate', '$timeout',
3.     '$filter', '$locale',
4.     function ($scope, utils, config, dao, $translate, $timeout, $filter, $locale) {
5.       // ----- initialisation modèle
6.       // modèle
7.       $scope.waiting = {text: config.msgWaiting, visible: false, cancel: cancel, time: 3000};
8.       $scope.server = {url: 'http://localhost:8080', login: 'admin', password: 'admin'};
9.       $scope.errors = {show: false, model: {}};
10.      $scope.medecins = {
11.        data: [
12.          {id: 1, version: 1, titre: "Mme", nom: "PELISSIER", prenom: "Marie"},
13.          {id: 2, version: 1, titre: "Mr", nom: "BROMARD", prenom: "Jacques"},
14.          {id: 3, version: 1, titre: "Mr", nom: "JANDOT", prenom: "Philippe"},
15.          {id: 4, version: 1, titre: "Melle", nom: "JACQUEMOT", prenom: "Justine"}
16.        ],
17.        title: config.listMedecins
18.      };
19.      var medecin = $scope.medecins.data[0];
20.      var clients = [
21.        {id: 1, version: 1, titre: "Mr", nom: "MARTIN", prenom: "Jules"},
22.        {id: 2, version: 1, titre: "Mme", nom: "GERMAN", prenom: "Christine"},


```

```

22.      {id: 3, version: 1, titre: "Mr", nom: "JACQUARD", prenom: "Maurice"},  

23.      {id: 4, version: 1, titre: "Melle", nom: "BISTROU", prenom: "Brigitte"}  

24.    ];  

25.    // locale française pour la date  

26.    angular.copy(config.locales['fr'], $locale);  

27.    var today = new Date();  

28.    var formattedDay = $filter('date')(today, 'yyyy-MM-dd');  

29.    var fullDay = $filter('date')(today, 'fullDate');  

30.    $scope.agenda = {title: config.agendaTitle, data: undefined, show: false, model: {titre:  
      médecin.titre, prenom: médecin.prenom, nom: médecin.nom, jour: fullDay}};  

31.  

32.  

33.    // ----- agenda initial  

34.    // la tâche asynchrone globale  

35.    var task;  

36.    // on demande l'agenda  

37.    getAgenda();  

38.  

39.    // ----- réservation  

40.    $scope.reserver = function (creneauId) {  

41. ....  

42.    };  

43.  

44.    // ----- suppression RV  

45.    $scope.supprimer = function (idRv) {  

46. ....  

47.    };  

48.  

49.    // obtention de l'agenda  

50.    function getAgenda() {  

51.    ...  

52.    }  

53.  

54.    // annulation attente  

55.    function cancel() {  

56. ....  

57.    }  

58.  ]);
```

- ligne 6 : configuration du message d'attente. Par défaut, on attendra 3 secondes avant de faire un appel HTTP ;
- ligne 7 : les informations nécessaires aux appels HTTP ;
- ligne 8 : configuration du message d'erreurs ;
- lignes 9-17 : les médecins en dur ;
- ligne 18 : un médecin particulier. C'est pour ses créneaux qu'on fera des réservations ;
- lignes 19-24 : les clients en dur ;
- ligne 26 : on veut manipuler des dates françaises ;
- ligne 27 : les rendez-vous seront pris à la date d'aujourd'hui ;
- ligne 28 : le service web de réservation attend des dates au format 'aaaa-mm-jj' ;
- ligne 29 : la date d'aujourd'hui sous la forme [jeudi 26 juin 2014] ;
- ligne 30 : configuration de l'agenda. L'attribut [model] transporte les paramètres du message internationalisé qui va être affiché :

```
  agenda_title: "Agenda de {{titre}} {{prenom}} {{nom}} le {{jour}}"
```

- ligne 35 : la variable globale [task] représente à un moment donné la tâche asynchrone en cours d'exécution ;
- ligne 37 : on demande l'agenda initial ;

C'est tout ce qui est fait lors du chargement initial de la page. Si tout se passe bien, la vue affiche l'agenda du jour de Mme PELISSIER.

Rdvmedecins - v1

Agenda de Mme Marie PELISSIER le jeudi 26 juin 2014

Créneau horaire	Client	Action
08h00:08h20	Mme Christine GERMAN	Supprimer
08h20:08h40		Réserver
08h40:09h00		Réserver
09h00:09h20	Mr Jules JACQUARD	Supprimer
09h20:09h40		Réserver
09h40:10h00		Réserver

3.7.9.4 Obtention de l'agenda

L'agenda est obtenu avec la méthode [getAgenda] suivante :

```
1. // obtention de l'agenda
2. function getAgenda() {
3.     // le chemin de l'URL de service
4.     var path = config.urlSrvAgenda + "/" + médecin.id + "/" + formattedDay;
5.     // on demande l'agenda
6.     task = dao.getData($scope.server.url, $scope.server.login, $scope.server.password, path);
7.     // msg d'attente
8.     $scope.waiting.visible = true;
9.     // on analyse le résultat de l'appel au service [dao]
10.    task.promise.then(function (result) {
11.        // fin d'attente
12.        $scope.waiting.visible = false;
13.        // erreur ?
14.        if (result.err == 0) {
15.            // on prépare le modèle de l'agenda
16.            $scope.agenda.data = result.data;
17.            $scope.agenda.show = true;
18.            // mise en forme de l'affichage des horaires
19.            angular.forEach($scope.agenda.data.creneauxMedecin, function (creneauMedecin) {
20.                creneauMedecin.creneau.text = utils.getTextForCreneau(creneauMedecin.creneau);
21.            });
22.        } else {
23.            // il y a eu des erreurs pour obtenir l'agenda
24.            $scope.errors = {title: config.getAgendaErrors, messages: utils.getErrors(result), show:
25.                true};
26.        }
27.    });
}
```

Ce code est celui étudié dans l'application précédente. Il y a deux changements :

- il n'y a pas d'attente simulée avant l'appel HTTP ;
- ligne 4 : on utilise le médecin créé lors de l'initialisation du contrôleur ainsi que le jour formaté qui a été construit ;

Ce code a été isolé dans une fonction car il est également utilisé par les fonctions [reserver] et [supprimer].

3.7.9.5 Réservation d'un créneau horaire

Créneau horaire	Client	Action	Créneau horaire	Client	Action
08h00:08h20	Mme Christine GERMAN	Supprimer	08h00:08h20	Mme Christine GERMAN	Supprimer
08h20:08h40		Réserver	08h20:08h40	Melle Brigitte BISTROU	Supprimer
08h40:09h00		Réserver	08h40:09h00		Réserver
09h00:09h20	Mr Jules JACQUARD	Supprimer	09h00:09h20	Mr Jules JACQUARD	Supprimer

On rappelle que les clients sont choisis de façon aléatoire.

Le code réservation est le suivant :

```

1. $scope.reserver = function (creneauId) {
2.     utils.debug("réservation du créneau", creneauId);
3.     // on crée un RV avec un client aléatoire dans le créneau identifié par [id]
4.     var idClient = clients[Math.floor(Math.random() * clients.length)].id;
5.     utils.debug("réservation du créneau pour le client", idClient);
6.     // attente simulée
7.     $scope.waiting.visible = true;
8.     var task = utils.waitForSomeTime($scope.waiting.time);
9.     // on ajoute le créneau
10.    var promise = task.promise.then(function () {
11.        // le chemin de l'URL de service
12.        var path = config.urlSrvRresaAdd;
13.        // les données à transmettre au service
14.        var post = {jour: formattedDay, idCreneau: creneauId, idClient: idClient};
15.        // on lance la tâche asynchrone
16.        task = dao.getData($scope.server.url, $scope.server.login, $scope.server.password,
    path, post);
17.        // on retourne la promesse d'achèvement de la tâche
18.        return task.promise;
19.    });
20.
21.    // analyse du résultat de la tâche
22.    promise = promise.then(function (result) {
23.        if (result.err != 0) {
24.            // il y a eu des erreurs pour valider le rv
25.            $scope.errors = {title: config.postResaErrors, messages: utils.getErrors(result,
    $filter), show: true};
26.        } else {
27.            // on demande le nouvel agenda
28.            getAgenda();
29.        }
30.    });
31.
32. };

```

- ligne 1 : on rappelle que le paramètre de la fonction [reserver] est le n° du créneau (attribut *id*) ;
- ligne 4 : un client est choisi de façon aléatoire dans la liste des clients définie en dur dans le code d'initialisation. On retient de lui son identifiant [id] ;
- lignes 7-8 : l'attente de 3 secondes ;
- lignes 11-18 : ces lignes ne sont exécutées qu'à la fin des 3 secondes ;
- ligne 12 : l'URL du service de réservation [/ajouterRv]. Cette URL est particulière vis à vis de celles qu'on a rencontrées jusqu'à maintenant. Elle est définie comme suit dans le service web :

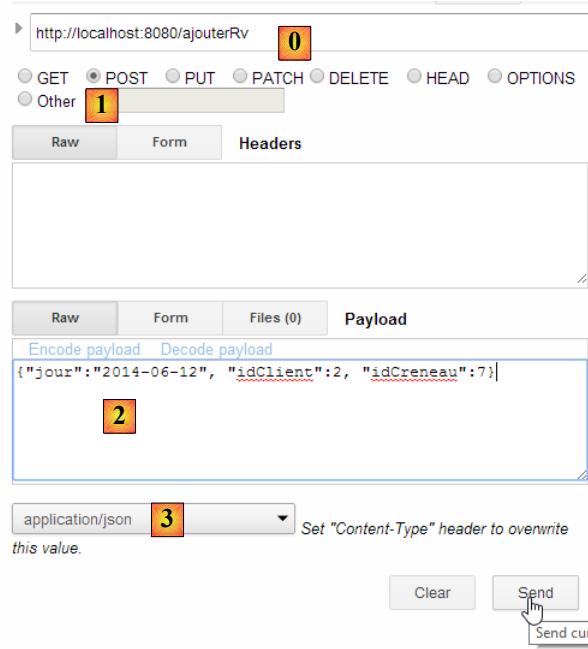
```

1.     @RequestMapping(value = "/ajouterRv", method = RequestMethod.POST, consumes =
    "application/json; charset=UTF-8")
2. public Reponse ajouterRv(@RequestBody PostAjouterRv post, HttpServletResponse response) {

```

- ligne 1 : l'URL n'a pas de paramètres et elle est demandée avec un POST ;
- ligne 2 : les paramètres postés le sont sous la forme d'un objet JSON. Celui-ci sera déserialisé dans le paramètre [post] (@RequestBody) ;

Nous avons vu un exemple de ce POST(page 74) :

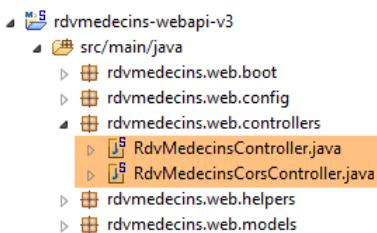


- en [0], l'URL du service web ;
- en [1], la méthode POST est utilisée ;
- en [2], le texte JSON des informations transmises au service web sous la forme {jour, idClient, idCreneau} ;
- en [3], le client précise au service web qu'il lui envoie des informations JSON ;

Revenons au code JS de la fonction [reserver] :

- ligne 14 : on crée la valeur à poster sous la forme d'un objet JS. Angular le sérialisera en JSON lorsqu'il sera posté ;
- ligne 16 : l'appel HTTP est fait. La valeur à poster est le dernier paramètre de la fonction [dao.getData]. Lorsque ce paramètre est présent, fonction [dao.getData] fait un POST au lieu d'un GET (revoir le code page 191, paragraphe 3.7.6.4) ;
- ligne 18 : on retourne la promesse de l'appel HTTP ;
- lignes 23-29 : ne sont exécutées que lorsque l'appel HTTP a rendu sa réponse ;
- ligne 23 : le paramètre [result] est de la forme [err,data] ou [err,messages] où [err] est un code d'erreur ;
- lignes 23-26 : s'il y a eu des erreurs, on rend visible le message d'erreur ;
- ligne 28 : si la réservation s'est bien passée, on réaffiche le nouvel agenda ;

3.7.9.6 Modification serveur



Dans la classe [RdvMedecinsCorsController], nous ajoutons la méthode suivante :

```
1.      // envoi des options au client
2.      private void sendOptions(HttpServletRequest response) {
```

```

3.     if (application.isCORSneeded()) {
4.         // on fixe le header CORS
5.         response.addHeader("Access-Control-Allow-Origin", "*");
6.         // on autorise le header [authorization]
7.         response.addHeader("Access-Control-Allow-Headers", "authorization");
8.     }
9.
10.    @RequestMapping(value = "/ajouterRv", method = RequestMethod.OPTIONS)
11.    public void ajouterRv(HttpServletRequest response) {
12.        sendOptions(response);
13.    }

```

L'ajout est fait lignes 10-13. Les entêtes des lignes 2-8 seront envoyés pour l'URL [/ajouterRv] (ligne 10) et la méthode HTTP [OPTIONS] (ligne 10).

La classe [RdvMedecinsController] est elle modifiée de la façon suivante :

```

1.     @RequestMapping(value = "/ajouterRv", method = RequestMethod.POST, consumes = "application/json;
  charset=UTF-8")
2.     public Reponse ajouterRv(@RequestBody PostAjouterRv post, HttpServletRequest response) {
3.         // entêtes CORS
4.         rdvMedecinsCorsController.ajouterRv(response);
5.     ...

```

Pour la méthode [POST] (ligne 1) et l'URL [/ajouterRv] (ligne 1), la méthode que nous venons d'ajouter dans [RdvMedecinsCorsController] est appelée (ligne 4), renvoyant donc les mêmes entêtes HTTP que pour la méthode HTTP [OPTIONS].

3.7.9.7 Tests

Faisons un premier test où nous réservons un créneau quelconque :

The screenshot shows the application interface. At the top, it says "Rdvmedecins - v1". Below that is a red error message box containing the text: "Les erreurs suivantes se sont produites lors de l'ajout du rendez-vous" followed by a bullet point: "L'accès au serveur n'a pu être réalisé. Vérifiez sa disponibilité". Below this is a blue header bar with the text "Agenda de Mme Marie PELISSIER le lundi 30 juin". The main area is a table with three columns: "Créneau horaire", "Client", and "Action". It contains two rows:

Créneau horaire	Client	Action
08h00:08h20		Réserver
08h20:08h40		Réserver

Comme toujours dans ces cas là, il faut regarder les logs de la console :

```

1. [dao] getData[/ajouterRv] error réponse : {"data": "", "status": 0, "config": {
  "method": "POST", "transformRequest": [null], "transformResponse": [
  null], "timeout": 1000, "url": "http://localhost:8080/ajouterRv", "data": {"jour": "2014-06-30", "idCreneau": 1, "idClient": 4}, "headers": {"Accept": "application/json, text/plain, */*", "Authorization": "Basic YWRtaW46YWRtaW4=", "Content-Type": "application/json; charset=utf-8"}, "statusText": ""}

```

La méthode [dao.getData] a échoué avec [status=0], ce qui signifie que c'est Angular qui a annulé la requête. On a la cause de l'erreur dans les logs :

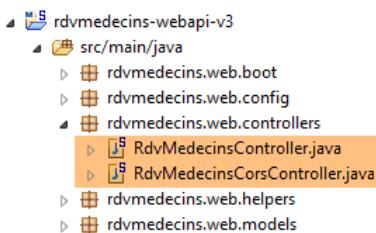
```
XMLHttpRequest cannot load http://localhost:8080/ajouterRv. Request header field Content-Type is not allowed by Access-Control-Allow-Headers.
```

Si on regarde les échanges réseau, on a la chose suivante :

Remote Address: ::1:8080
 Request URL: http://localhost:8080/ajouterRv
Request Method: OPTIONS 2
 Status Code: 200 OK
Request Headers view source
 Accept: */*
 Accept-Encoding: gzip,deflate,sdch
 Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
Access-Control-Request-Headers: accept, authorization, content-type 3
 Access-Control-Request-Method: POST 3
 Cache-Control: no-cache
 Connection: keep-alive
 Host: localhost:8080
 Origin: http://localhost:63342
 Pragma: no-cache
 Referer: http://localhost:63342/rdvmedecins-angular-v1/app-19.html
 User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36
Response Headers view source
 Access-Control-Allow-Headers: authorization 4
 Access-Control-Allow-Origin: *
 Allow: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, PATCH
 Cache-Control: no-cache, no-store, max-age=0, must-revalidate
 Content-Length: 0

- en [1] et [2] : il n'y a eu qu'une requête HTTP, la requête [OPTIONS] ;
- en [3], le client Angular demande deux autorisations :
 - celle d'envoyer les entêtes HTTP [accept, authorization, content-type] ;
 - celle d'envoyer une commande POST ;
- en [4] : le serveur autorise l'entête [authorization]. Rappelons que côté serveur, c'est nous-mêmes qui envoyons cette autorisation ;

La nouveauté est donc que sur une opération POST, le client Angular demande davantage d'autorisations au serveur. Il faut donc modifier celui-ci pour qu'il les lui donne :



Dans la classe [RdvMedecinsCorsController], nous modifions la méthode privée qui génère les entêtes HTTP envoyés pour les commandes OPTIONS, GET et POST :

```

1.     // envoi des options au client
2.     private void sendOptions(HttpServletRequest response) {
3.         if (application.isCORSneeded()) {
4.             // on fixe le header CORS
5.             response.addHeader("Access-Control-Allow-Origin", "*");
6.             // on autorise certains headers
7.             response.addHeader("Access-Control-Allow-Headers", "accept, authorization, content-
   type");
8.             // on autorise le POST
9.             response.addHeader("Access-Control-Allow-Methods", "POST");
10.        }
11.    }

```

- ligne 7 : on a ajouté une autorisation pour les entêtes HTTP [accept, content-type] ;

- ligne 9 : on a ajouté une autorisation pour la méthode POST ;

On refait le test après avoir relancé le serveur :

Rdvmedecins - v1

Agenda de Mme Marie PELISSIER le lundi 30 juin

Créneau horaire	Client	Action
08h00:08h20	Mr Jules MARTIN	Supprimer
08h20:08h40		Réserver

Cette fois-ci, on a réussi à réserver.

3.7.9.8 Suppression d'un rendez-vous

Créneau horaire	Client	Action	Créneau horaire	Client	Action
08h00:08h20	Mme Christine GERMAN	Supprimer	08h00:08h20		Réserver
08h20:08h40		Réserver	08h20:08h40		Réserver
08h40:09h00		Réserver	08h40:09h00		Réserver

Le code de la fonction [supprimer] est le suivant :

```

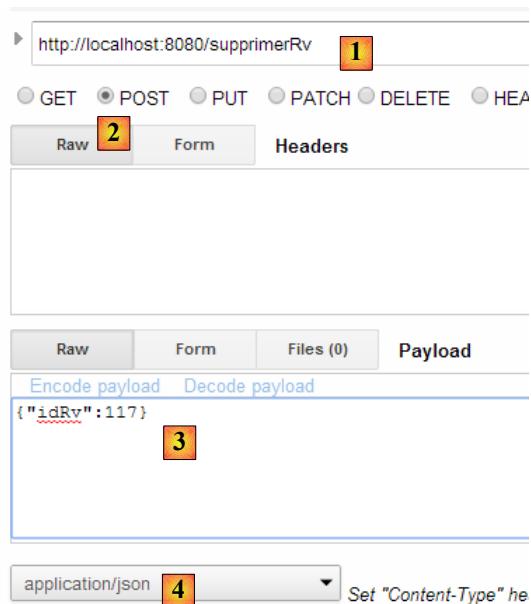
1. $scope.supprimer = function (idRv) {
2.     utils.debug("suppression rv n°", idRv);
3.     // attente simulée
4.     $scope.waiting.visible = true;
5.     task = utils.waitForSomeTime($scope.waiting.time);
6.     // on supprime le rendez-vous
7.     var promise = task.promise.then(function () {
8.         // le chemin de l'URL de service
9.         var path = config.urlSvrResaRemove;
10.        // les données à transmettre au service
11.        var post = {idRv: idRv};
12.        // on lance la tâche asynchrone
13.        task = dao.getData($scope.server.url, $scope.server.login, $scope.server.password, path, post);
14.        // on retourne la promesse d'achèvement de la tâche
15.        return task.promise;
16.    });
17.
18.    // analyse du résultat de la tâche
19.    promise = promise.then(function (result) {
20.        if (result.err != 0) {
21.            // il y a eu des erreurs pour supprimer le rv
22.            $scope.errors = {title: config.postRemoveErrors, messages: utils.getErrors(result, $filter),
show: true};
23.            // on met à jour l'UI
24.            $scope.waiting.visible = false;
25.        } else {
26.            // on demande le nouvel agenda
27.            getAgenda();
28.        }
29.    });
30. };

```

- ligne 1 : il faut se rappeler que le paramètre de la fonction est le n° du rendez-vous à supprimer. On a là un code très similaire à celui de la réservation. Nous ne commentons que les différences ;
- ligne 9 : l'URL du service est ici [/supprimerRv] et là également elle est accédée via un POST :

```
1. @RequestMapping(value = "/supprimerRv", method = RequestMethod.POST, consumes =
"application/json; charset=UTF-8")
2. public Reponse supprimerRv(@RequestBody PostSupprimerRv post, HttpServletReponse response) {
```

Le paramètre posté est là encore transmis sous une forme JSON. A la page 99, nous avons montré la nature du POST réalisé à la main :



- en [1], l'URL du service web ;
- en [2], la méthode POST est utilisée ;
- en [3], le texte JSON des informations transmises au service web sous la forme {idRv} ;
- en [4], le client précise au service web qu'il lui envoie des informations JSON ;

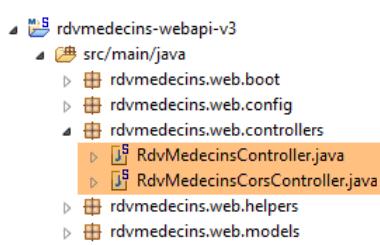
Revenons au code JS de la fonction [supprimer] :

- ligne 11 : on crée l'objet posté. Angular le sérialisera automatiquement en JSON ;

Le reste du code est analogue à celui de la réservation.

3.7.9.9 Modification serveur

Côté serveur, nous faisons les modifications suivantes :



Dans la classe [RdvMedecinsCorsController], nous ajoutons la méthode suivante :

```

1.    // envoi des options au client
2.    private void sendOptions(HttpServletRequest response) {
3.        if (application.isCORSneeded()) {
4.            // on fixe le header CORS
5.            response.addHeader("Access-Control-Allow-Origin", "*");
6.            // on autorise certains headers
7.            response.addHeader("Access-Control-Allow-Headers", "accept, authorization, content-
8.                type");
9.            // on autorise le POST
10.           response.addHeader("Access-Control-Allow-Methods", "POST");
11.       }
12.   ...
13.   @RequestMapping(value = "/supprimerRv", method = RequestMethod.OPTIONS)
14.   public void supprimerRv(HttpServletRequest response) {
15.       sendOptions(response);
16.   }

```

L'ajout est fait lignes 13-16. Les entêtes des lignes 2-10 seront envoyés pour l'URL [/supprimerRv] (ligne 13) et la méthode HTTP [OPTIONS] (ligne 13).

La classe [RdvMedecinsController] est elle, modifiée de la façon suivante :

```

1.    @RequestMapping(value = "/supprimerRv", method = RequestMethod.POST, consumes = "application/json;
2.                      charset=UTF-8")
3.    public Reponse supprimerRv(@RequestBody PostSupprimerRv post, HttpServletRequest response) {
4.        // entêtes CORS
5.        rdvMedecinsCorsController.supprimerRv(response);
5.    ...

```

Pour la méthode [POST] (ligne 1) et l'URL [/supprimerRv] (ligne 1), la méthode que nous venons d'ajouter dans [RdvMedecinsCorsController] est appelée (ligne 4), renvoyant donc les mêmes entêtes HTTP que pour la méthode HTTP [OPTIONS].

3.7.10 Exemple 10 : créer et annuler des réservations - 2

Nous présentons maintenant la même application que précédemment mais au lieu de réserver pour un client aléatoire, celui-ci sera sélectionné dans une liste déroulante.

3.7.10.1 La vue V de l'application

Nous présenterons le formulaire suivant :

Rdvmédecins - v1

The screenshot shows a web application interface. At the top, it says "Agenda de Mme Marie PELISSIER le jeudi 26 juin 2014". Below that is a section titled "Liste des clients". A dropdown menu shows "Melle Brigitte BISTROU" and a red square button with the number "1". The main area displays a table with four rows:

Créneau horaire	Client	Action
08h00:08h20	Melle Brigitte BISTROU	Supprimer
08h20:08h40		Réserver
08h40:09h00		Réserver

Les clients seront sélectionnés en [1].

Le code est similaire à celui de l'application précédente, aussi ne présentons-nous que les principales différences.

Nous dupliquons le fichier [app-19.html] dans [app-20.html] puis nous créons le code de la liste déroulante des clients [1] :

```
1. <!-- la liste des clients -->
2.   <div class="alert alert-info">
3.     <h3>{{agenda.title|translate:agenda.model}}</h3>
4.
5.   <div class="row" ng-show="clients.show">
6.     <div class="col-md-3">
7.       <h2 translate="{{clients.title}}></h2>
8.       <select data-style="btn-primary" class="selectpicker" select-enable="" ng-if="clients.data">
9.         <option ng-repeat="client in clients.data" value="{{client.id}}>
10.           {{client.titre}} {{client.prenom}} {{client.nom}}
11.         </option>
12.       </select>
13.     </div>
14.   </div>
15. </div>
```

- lignes 8-12 : la liste déroulante va être implémentée avec le composant [bootstrap-select] ;
- ligne 1 : la directive [selectEnable] est appliquée via l'attribut [select-enable] ;
- ligne 1 : la balise <select> n'est générée que si [clients.data] existe (# null, undefined). Ce point est important et a été expliqué page 215;

Par ailleurs, nous importons de nouveaux fichiers JS :

```
1. <script type="text/javascript" src="rdvmédecins-08.js"></script>
2. <!-- directives -->
3. <script type="text/javascript" src="selectEnable.js"></script>
4. <script type="text/javascript" src="footable.js"></script>
```

- ligne 1 : le fichier [rdvmédecins-08.js] est obtenu par recopie du fichier [rdvmédecins-0.js] ;
- lignes 3-4 : on importe les fichiers des deux directives ;

3.7.10.2 Le contrôleur C

Le code du contrôleur C évolue de la façon suivante :

```

1. // contrôleur
2. angular.module("rdvmedecins")
3.   .controller('rdvMedecinsCtrl', ['$scope', 'utils', 'config', 'dao', '$translate', '$timeout', '$filter',
4.   '$locale',
5.     function ($scope, utils, config, dao, $translate, $timeout, $filter, $locale) {
6.     // ----- initialisation modèle
7.     ...
8.     $scope.clients = {title: config.listClients, show: false, model: {}};
9.
10.    //----- initialisation vue
11.    // la tâche asynchrone globale
12.    var task;
13.    // on demande les clients puis l'agenda
14.    getClients().then(function () {
15.      getAgenda();
16.    });
17.    ...
18.
19.    // exécution action
20.    function getClients() {
21.      ...
22.    };
23.  }]);

```

- ligne 8 : l'objet [\$scope.clients] configure la liste déroulante des clients dans la vue V ;
- lignes 14-16 : de façon asynchrone, on demande d'abord la liste des clients, puis une fois celle-ci obtenue, on demande l'agenda de Mme PELISSIER pour le jour d'aujourd'hui. La syntaxe utilisée ici ne fonctionne que parce que la fonction [getClients] rend une promesse (promise) ;

La méthode [getClients] demande la liste des clients :

```

1. function getClients() {
2.   // on met à jour l'UI
3.   $scope.waiting.visible = true;
4.   $scope.clients.show = false;
5.   $scope.errors.show = false;
6.   // on demande la liste des clients;
7.   task = dao.getData($scope.server.url, $scope.server.login, $scope.server.password,
  config.urlSrvClients);
8.   var promise = task.promise;
9.   // on analyse le résultat de l'appel précédent
10.  promise = promise.then(function (result) {
11.    // result={err: 0, data: [client1, client2, ...]}
12.    // result={err: n, messages: [msg1, msg2, ...]}
13.    if (result.err == 0) {
14.      // on met les données acquises dans le modèle
15.      $scope.clients.data = result.data;
16.      // on met à jour l'UI
17.      $scope.clients.show = true;
18.      $scope.waiting.visible = false;
19.    } else {
20.      // il y a eu des erreurs pour obtenir la liste des clients
21.      $scope.errors = { title: config.getClientsErrors, messages: utils.getErrors(result),
        show: true, model: {}};
22.      // on met à jour l'UI
23.      $scope.waiting.visible = false;
24.    }
25.  });
26.  // on rend la promesse
27.  return promise;
28. };

```

C'est un code que nous avons déjà rencontré et commenté. L'élément important à noter est ligne 31 :

- ligne 27 : on rend la promesse de la ligne 10, c-à-d la dernière promesse obtenue dans le code. Cette promesse ne sera obtenue que lorsque l'appel HTTP aura rendu sa réponse ;

La méthode [reserver] évolue légèrement :

```

1.     $scope.reserver = function (creneauId) {
2.         utils.debug("réservation du créneau", creneauId);
3.         // on crée un RV pour le client sélectionné
4.         var idClient = $(".selectpicker").selectpicker('val');
5.         ...
6.     });

```

- ligne 4 : on ne réserve plus pour un client aléatoire mais pour le client sélectionné dans la liste des clients.

3.7.11 Exemple 11 : une directive [selectEnable2]

Cet exemple revient sur les directives.

3.7.11.1 La vue V

L'application affiche la vue suivante :

The screenshot shows a web application interface titled "Rdvmedecins - v1". It consists of two main sections. The top section is titled "Liste des clients" and contains a dropdown menu with the value "Mr Jules MARTIN". The bottom section is titled "Liste des médecins" and also contains a dropdown menu with the value "Mme Marie PELISSIER". Both sections have a blue header bar and a light blue background.

3.7.11.2 Le code HTML de la vue

Le code HTML de la vue [app-21.html] est le suivant :

```

1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <!-- le message d'attente -->
5.   <div class="alert alert-warning" ng-show="waiting.visible">
6.     ...
7.   </div>
8.
9.   <!-- la liste d'erreurs -->
10.  <div class="alert alert-danger" ng-show="errors.show">
11.    ...
12.  </div>
13.
14.  <!-- la liste des clients -->
15.  <div class="alert alert-info">
16.    <div class="row" ng-show="clients.show">
17.      <div class="col-md-4">
18.        <h2 translate="{{clients.title}}></h2>
19.        <select data-style="btn-primary" id="selectpickerClients" select-enable2="" ng-if="clients.data">
20.          <option ng-repeat="client in clients.data" value="{{client.id}}>
21.            {{client.titre}} {{client.prenom}} {{client.nom}}
22.          </option>

```

```

23.      </select>
24.    </div>
25.  </div>
26. </div>
27.
28.  <!-- la liste des médecins -->
29. <div class="alert alert-info">
30.   <div class="row" ng-show="medecins.show">
31.     <div class="col-md-4">
32.       <h2 translate="{{medecins.title}}"></h2>
33.       <select data-style="btn-primary" id="selectpickerMedecins" select-enable2="" ng-
  if="medecins.data">
34.         <option ng-repeat="medecin in medecins.data" value="{{medecin.id}}>
35.           {{medecin.titre}} {{medecin.prenom}} {{medecin.nom}}
36.         </option>
37.       </select>
38.     </div>
39.   </div>
40. </div>
41. </div>
42. ...
43. <script type="text/javascript" src="rdvmedecins-09.js"></script>
44. <!-- directives -->
45. <script type="text/javascript" src="selectEnable2.js"></script>

```

- lignes 19-23 : la liste déroulante des clients ;
- ligne 19 : on applique la directive [selectEnable2] (attribut [select-enable2]) ;
- ligne 19 : que si [clients.data] est non vide ;
- ligne 19 : la liste déroulante est identifiée par l'attribut [id="selectpickerClients"] ;
- lignes 33-37 : la liste déroulante des médecins ;
- ligne 33 : on applique la directive [selectEnable2] (attribut [select-enable2]) ;
- ligne 33 : que si [medecins.data] est non vide ;
- ligne 33 : la liste déroulante est identifiée par l'attribut [id="selectpickerMedecins"] ;
- ligne 43 : on importe un nouveau fichier JS [[rdvmedecins-09.js](#)] ;
- ligne 45 : on importe le fichier JS de la nouvelle directive ;

3.7.11.3 La directive [selectEnable2]

Le code de la directive [selectEnable2] est le suivant :

```

1. angular.module("rdvmedecins").directive('selectEnable2', ['$timeout', 'utils', function ($timeout, utils) {
2.   return {
3.     link: function (scope, element, attrs) {
4.       utils.debug("directive selectEnable2 attrs", attrs);
5.       $timeout(function () {
6.         $('#' + attrs['id']).selectpicker();
7.       })
8.     }
9.   }
10. }]);

```

- ligne 4 : on fait afficher la valeur du paramètres [attrs] afin de faire comprendre le fonctionnement du code. On va découvrir que attrs['id']='selectpickerClients' pour la liste des clients ;
- ligne 6 : pour localiser dans le DOM un élément d'[id=x'], on écrit \${'#x'}. Donc on doit écrire \${'#selectpickerClients'} pour localiser la liste des clients. Ceci est obtenu avec la syntaxe [\${'#' + attrs['id']}];

La directive [selectEnable2] utilise donc l'information transportée par l'un des attributs de l'élément HTML auquel elle est appliquée.

3.7.11.4 Le contrôleur C

Le contrôleur C se trouve dans le fichier JS [[rdvmedecins-09.js](#)] et a la structure suivante :

```

1. // contrôleur
2. angular.module("rdvmedecins")
3.   .controller('rdvMedecinsCtrl', ['$scope', 'utils', 'config', 'dao',
4.     function ($scope, utils, config, dao) {

```

```

5.      // ----- initialisation modèle
6.      // le msg d'attente
7.      $scope.waiting = {text: config.msgWaiting, visible: false, cancel: cancel, time: 3000};
8.      // les informations de connexion
9.      $scope.server = {url: 'http://localhost:8080', login: 'admin', password: 'admin'};
10.     // les erreurs
11.     $scope.errors = {show: false, model: {}};
12.     // les médecins
13.     $scope.medecins = {title: config.listMedecins, show: false, model: {}};
14.     // les clients
15.     $scope.clients = {title: config.listClients, show: false, model: {}};
16.
17.     // la tâche asynchrone globale
18.     var task;
19.     // ----- initialisation vue
20.     // on met à jour l'UI
21.     $scope.waiting.visible = true;
22.     $scope.clients.show = false;
23.     $scope.medecins.show = false;
24.     $scope.errors.show = false;
25.     // on demande les clients puis les médecins
26.     getClients().then(function () {
27.         getMedecins();
28.     });
29.
30.     // liste des clients
31.     function getClients() {
32.         ...
33.     }
34.
35.     // liste des médecins
36.     function getMedecins() {
37.     ...
38.     }
39.
40.     // annulation attente
41.     function cancel() {
42.     ...
43.     }
44. });

```

- lignes 26-28 : on demande d'abord les clients puis les médecins ;

3.7.11.5 Les tests

Testez cette nouvelle version.

3.7.12 Exemple 12 : une directive [list]

Nous reprenons le même exemple que précédemment mais nous voulons alléger le code HTML en utilisant une directive. En effet, nous avons actuellement le code HTML suivant :

```

1. <!-- la liste des clients -->
2. <div class="alert alert-info">
3.   <div class="row" ng-show="clients.show">
4.     <div class="col-md-4">
5.       <h2 translate="{{clients.title}}></h2>
6.       <select data-style="btn-primary" id="selectpickerClients" select-enable2="" ng-if="clients.data">
7.         <option ng-repeat="client in clients.data" value="{{client.id}}>
8.           {{client.titre}} {{client.prenom}} {{client.nom}}
9.         </option>
10.        </select>
11.      </div>
12.    </div>
13.  </div>
14. <!-- la liste des médecins -->
15. <div class="alert alert-info">
16.   <div class="row" ng-show="medecins.show">
17.     <div class="col-md-4">
18.       <h2 translate="{{medecins.title}}></h2>
19.       <select data-style="btn-primary" id="selectpickerMedecins" select-enable2="" ng-
if="medecins.data">
20.         <option ng-repeat="medecin in medecins.data" value="{{medecin.id}}>

```

```

21.           {{medecin.titre}} {{medecin.prenom}} {{medecin.nom}}
22.       </option>
23.     </select>
24.   </div>
25. </div>
26. </div>

```

Les lignes 14-26 sont identiques aux lignes 1-13. Elles s'appliquent à des médecins au lieu des clients. Nous voudrions pouvoir écrire la chose suivante :

```

1.  <!-- la liste des clients -->
2.  <list model="clients" ng-if="clients.show"></list>
3.  <!-- la liste des médecins -->
4.  <list model="medecins" ng-if="medecins.show"></list>

```

Ce code fait intervenir une nouvelle directive [list] que nous allons créer maintenant.

3.7.12.1 La directive [list]

La directive [list] est placée dans le fichier JS [list.js]. Son code est le suivant :

```

1. angular.module("rdvmedecins")
2.   .directive("list", ['$utils', '$timeout', function (utils, $timeout) {
3.     // instance de la directive retournée
4.     return {
5.       // élément HTML
6.       restrict: "E",
7.       // url du fragment
8.       templateUrl: "list.html",
9.       // scope unique à chaque instance de la directive
10.      scope: true,
11.      // fonction lien avec le document
12.      link: function (scope, element, attrs) {
13.        utils.debug("directive list attrs", attrs);
14.        scope.model = scope[attrs['model']];
15.        utils.debug("directive list model", scope.model);
16.        $timeout(function () {
17.          $('#' + scope.model.id).selectpicker();
18.        })
19.      }
20.    }
21.  }]);

```

- ligne 2 : définit une directive nommée 'list' ;
- ligne 6 : l'attribut [restrict] fixe les façons d'utiliser la directive. [restrict: "E"] signifie que la directive [list] est utilisable comme élément HTML <list ...>...</list>. [restrict: "A"] signifie que la directive [list] est utilisable comme attribut, par exemple <div ... list='...'>. [restrict: "AE"] signifie que la directive [list] est utilisable comme attribut et comme élément ;
- ligne 8 : l'attribut [templateUrl] indique le nom du fragment HTML à utiliser à la rencontre de la balise. Ce fragment sera le corps de la balise ;
- ligne 10 : l'attribut [scope] fixe la portée du modèle de la directive. [scope: true] signifie que deux éléments de type <list> auront chacun leur modèle. Par défaut, (scope non initialisé), ils partagent leurs modèles ;
- ligne 12 : la fonction [link] que nous avons utilisée déjà plusieurs fois ;

Pour comprendre le code ci-dessus, il faut se rappeler l'utilisation qui va être faite de la directive :

```

1.  <!-- la liste des clients -->
2.  <list model="clients" ng-if="clients.show"></list>
3.  <!-- la liste des médecins -->
4.  <list model="medecins" ng-if="medecins.show"></list>

```

La directive [list] est utilisée comme élément HTML <list>. Cet élément a deux attributs :

- [model] : qui va avoir pour valeur, l'élément du modèle M de la vue V dans laquelle se trouve la directive [list]. Cet élément va alimenter le modèle de la directive ;
- [ng-if] : qui va faire en sorte que le code HTML de la directive ne soit pas généré s'il n'y a rien à visualiser ;

Revenons au code de la fonction [link] de la directive :

```

1. link: function (scope, element, attrs) {
2.   utils.debug("directive list attrs", attrs);

```

```

3.     scope.model = scope[attrs['model']];
4.     utils.debug("directive list model", scope.model);
5.     $timeout(function () {
6.       $('#' + scope.model.id).selectpicker();
7.     })
8.   }

```

Associons ce code JS avec le code HTML qui utilise la directive :

```
<list model="clients" ng-if="clients.show"></list>
```

- ligne 3 : `attrs['model']` a ici pour valeur 'clients' ;
- ligne 3 : `scope[attrs['model']]` a pour valeur `scope['clients']` et représente alors `[$scope.clients]`, c-à-d le champ [clients] du modèle de la vue. Ce champ aura pour valeur `{id :..., data:[client1, client2, ...], show : ..., title :...}` ;
- ligne 3 : on ajoute un champ [model] au modèle de la directive. Celle-ci a hérité du modèle de la vue dans laquelle elle se trouve. Il faut donc éviter les collisions avec un éventuel champ [model] que pourrait avoir également la vue. Ici, il n'y a pas de collision ;
- ligne 4 : on affiche [scope.model] afin de mieux comprendre le code ;
- lignes 5-7 : on retrouve un code déjà rencontré. La différence est que l'`id` du composant était pris auparavant dans un attribut `attrs['id']`. Là il sera pris dans `[scope.model.id]` ;

Maintenant, regardons le code HTML généré par la directive. A cause de l'attribut `[templateUrl: "list.html"]` de la directive, il faut le chercher dans le fichier `[list.html]` :

```

1. <!-- une liste de clients ou de médecins -->
2. <div class="alert alert-info" ng-show="model.show">
3.   <div class="row">
4.     <div class="col-md-4">
5.       <h2 translate="{{model.title}}></h2>
6.       <select data-style="btn-primary" id="{{model.id}}" ng-if="model.data">
7.         <option ng-repeat="element in model.data" value="{{element.id}}>
8.           {{element.titre}} {{element.prenom}} {{element.nom}}
9.         </option>
10.        </select>
11.      </div>
12.    </div>
13.  </div>

```

- la première chose qu'il faut se rappeler pour lire ce code est que la directive a créé un objet `[scope.model]` de la forme `[{id :..., data:[client1, client2, ...], show : ..., title :...}]`. Cet objet [model] (`scope` est implicite dans le code HTML) est utilisé par le code HTML de la directive ;
- ligne 2 : utilisation de `[model.show]` pour montrer / cacher la vue générée par la directive ;
- ligne 5 : utilisation de `[model.title]` pour mettre un titre ;
- ligne 6 : utilisation de `[model.id]` pour mettre un id à la balise `<select>`. Cet `id` est utilisé par le code JS de la directive ;
- ligne 6 : utilisation de `[model.data]` pour générer le `<select>` uniquement s'il y a des données à afficher ;
- lignes 7-9 : utilisation de `[model.data]` pour générer les éléments de la liste déroulante ;

3.7.12.2 Le code HTML

Le code HTML de l'application `[app-22.html]` est le suivant :

```

1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <!-- le message d'attente -->
5.   <div class="alert alert-warning" ng-show="waiting.visible">
6.     ...
7.   </div>
8.
9.   <!-- la liste d'erreurs -->
10.  <div class="alert alert-danger" ng-show="errors.show">
11.    ...
12.  </div>
13.
14.  <!-- la liste des clients -->
15.  <list model="clients" ng-if="clients.show"></list>
16.  <!-- la liste des médecins -->
17.  <list model="medecins" ng-if="medecins.show"></list>
18. </div>

```

```

19. ...
20. <script type="text/javascript" src="rdvmedecins-10.js"></script>
21. <!-- directives -->
22. <script type="text/javascript" src="list.js"></script>

```

- ligne 22 : il ne faut pas oublier d'inclure le code JS de la directive ;

3.7.12.3 Le contrôleur C

Le contrôleur C évolue très peu :

```

1. angular.module("rdvmedecins")
2.   .controller('rdvMedecinsCtrl', ['$scope', 'utils', 'config', 'dao',
3.     function ($scope, utils, config, dao) {
4.       // ----- initialisation modèle
5.     ...
6.       // les médecins
7.       $scope.medecins = {title: config.listMedecins, show: false, id: 'medecins'};
8.       // les clients
9.       $scope.clients = {title: config.listClients, show: false, id: 'clients'};
10.    ...

```

- lignes 7 et 9, nous ajoutons l'attribut [id] aux modèles des médecins et des clients ;

3.7.12.4 Les tests

Les tests donnent les mêmes résultats que dans l'exemple précédent.

3.7.13 Exemple 13 : mise à jour du modèle d'une directive

Nous restons dans l'étude des directives et nous gardons l'exemple de la liste déroulante. On veut étudier ici le comportement de la directive [list] lorsque le contenu de la liste déroulante change.

3.7.13.1 Les vues V

Les différentes vues sont les suivantes :

The screenshot displays two views of the application. The left view shows a button labeled "Liste des clients" with a red box around it and the number "1" below it. The right view shows a dropdown menu titled "Liste des clients" containing four items: "Mr Jules MARTIN", "Mme Christine GERMAN", "Mr Jules JACQUARD", and "Melle Brigitte BISTRO". The first item is highlighted.

- en [1], on demande une première fois la liste des clients ;

Rdvmedecins - v1



2

Rdvmedecins - v1



Liste des clients



- en [2], on demande une seconde fois la liste des clients. Cette seconde liste est alors cumulée à la première [3]. C'est la mise à jour du composant [Bootstrap select] qu'on veut étudier dans cet exemple.

3.7.13.2 La page HTML

La page HTML [app-23.html] est obtenue par recopie de [app-22.html] puis modifiée de la façon suivante :

```
1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <!-- le message d'attente -->
5.   <div class="alert alert-warning" ng-show="waiting.visible">
6.     ...
7.   </div>
8.
9.   <!-- la liste d'erreurs -->
10.  <div class="alert alert-danger" ng-show="errors.show">
11.    ...
12.  </div>
13.
14.  <!-- le bouton -->
15.  <div class="alert alert-warning">
16.    <button class="btn btn-primary" ng-click="getClients()">{{clients.title|translate}}</button>
17.  </div>
18.
19.  <!-- la liste des clients -->
20.  <list2 model="clients" ng-if="clients.show"></list2>
21. </div>
22. ...
23. <script type="text/javascript" src="rdvmedecins-11.js"></script>
24. <!-- directives -->
25. <script type="text/javascript" src="list2.js"></script>
```

Les modifications par rapport à l'application précédente sont les suivantes :

- lignes 15-17 : ajout d'un bouton ;
- ligne 20 : utilisation d'une nouvelle directive [list2] ;
- ligne 23 : utilisation d'un nouveau fichier JS ;
- ligne 25 : importation du fichier JS de la directive [list2] ;

3.7.13.3 La directive [list2]

La directive [list2] dans [list2.js] est la suivante :

```
1. angular.module("rdvmedecins")
2.   .directive("list2", ['$utils', '$timeout', function (utils, $timeout) {
3.     // instance de la directive retournée
4.     return {
5.       // élément HTML
6.       restrict: "E",
7.       // url du fragment
8.       templateUrl: "list.html",
9.       // scope unique à chaque instance de la directive
10.      scope: true,
11.      // fonction lien avec le document
12.      link: function (scope, element, attrs) {
13.        utils.debug('directive list2');
14.        scope.model = scope[attrs['model']];
15.        $timeout(function () {
16.          $('#' + scope.model.id).selectpicker('refresh');
17.        })
18.      }
19.    }
20.  }]);

```

La seule différence avec la directive [list] est ligne 16 : avec la méthode [selectpicker('refresh')], on demande au composant [Bootstrap-select] de se rafraîchir. L'idée derrière cela est qu'à chaque fois que l'utilisateur va demander une nouvelle liste de clients, on va rafraîchir la liste déroulante. Ca ne va pas marcher mais c'est l'idée de base.

3.7.13.4 Le contrôleur C

Le contrôleur est dans le fichier [[rdvmedecins-11.js](#)] obtenu par recopie du fichier [[rdvmedecins-10.js](#)] :

```
1.      // les clients
2.      $scope.clients = {title: config.listClients, show: false, id: 'clients', data: []};
3. ...
4.      // liste des clients
5.      $scope.getClients = function getClients() {
6.        // on met à jour l'UI
7.        $scope.waiting.visible = true;
8.        $scope.errors.show = false;
9.        // on demande la liste des clients;
10.       task = dao.getData($scope.server.url, $scope.server.login, $scope.server.password,
config.urlSrvClients);
11.       var promise = task.promise;
12.       // on analyse le résultat de l'appel précédent
13.       promise = promise.then(function (result) {
14.         // result={err: 0, data: [client1, client2, ...]}
15.         // result={err: n, messages: [msg1, msg2, ...]}
16.         if (result.err == 0) {
17.           // on met les données acquises dans un nouveau modèle pour forcer la vue à se rafraîchir
18.           $scope.clients = {title: $scope.clients.title, data: $scope.clients.data.concat(result.data),
show: $scope.clients.show, id: $scope.clients.id};
19.           // on met à jour l'UI
20.           $scope.clients.show = true;
21.           $scope.waiting.visible = false;
22.         } else {
23.           // il y a eu des erreurs pour obtenir la liste des clients
24.           $scope.errors = { title: config.getClientsErrors, messages: utils.getErrors(result), show:
true, model: {}};
25.           // on met à jour l'UI
26.           $scope.waiting.visible = false;
27.         }
28.       });
29.     }

```

- ligne 1 : afin de permettre la concaténation de tableaux dans [clients.data], cet objet est initialisé avec un tableau vide ;
- ligne 18 : on concatène la nouvelle liste de clients avec celles déjà présentes dans le tableau [clients.data] ;

Avant on avait écrit :

```
1. // on met les données acquises dans le modèle
2. $scope.clients.data = result.data;
```

Maintenant on écrit :

```
1. // on met les données acquises dans un nouveau modèle pour forcer la vue à se rafraîchir
2. $scope.clients = {title: $scope.clients.title, data: $scope.clients.data.concat(result.data), show: $scope.clients.show, id: $scope.clients.id};
```

Pour comprendre ce code, il faut se rappeler comment le modèle M est utilisé dans la vue V dans le cas de la directive [list2] :

```
1. <!-- la liste des clients -->
2. <list2 model="clients" ng-if="clients.show"></list2>
```

Le modèle utilisé par la directive [list2] est [clients]. Elle ne sera réévaluée dans la vue V, que si [clients] change dans le modèle M de la vue. La première idée qui vient pour la modification est d'écrire :

```
$scope.clients.data=$scope.clients.data.concat(result.data) ;
```

pour tenir compte du fait que la nouvelle liste de clients doit être ajoutée aux précédentes. Ce faisant, on modifie [clients.data] mais pas [clients]. Je ne connais pas les arcanes de Javascript mais il ne serait pas étonnant que [clients] soit un pointeur, ainsi que [clients.data]. Le pointeur [clients] ne change pas lorsqu'on change le pointeur [clients.data]. La directive [list2] n'est alors pas réévaluée. C'est effectivement ce qu'on constate lorsqu'on débogue l'application (F12 dans Chrome).

En écrivant :

```
$scope.clients = {title: $scope.clients.title, data: $scope.clients.data.concat(result.data), show: $scope.clients.show, id: $scope.clients.id};
```

On s'assure que [\$scope.clients] reçoit bien une nouvelle valeur. Le pointeur [\$scope.clients] pointe sur un nouvel objet. La directive [list2] devrait alors être réévaluée. Mais pourtant, on n'a pas le résultat cherché. Examinons les copies d'écran lorsqu'on demande deux fois la liste des clients :

Rdvmedecins - v1

Liste des clients

Liste des clients

Mr Jules MARTIN



- 1 Mr Jules MARTIN
- Mme Christine GERMAN
- Mr Jules JACQUARD
- Melle Brigitte BISTROU

```
<h2 translate="msg_list_clients" class="ng-scope">Liste des clients</h2>
<!-- ngIf: model.data -->
<select data-style="btn-primary" id="clients" ng-if="model.data" class="ng-scope" style="display: none;">
  <!-- ngRepeat: element in model.data -->
  <option ng-repeat="element in model.data" value="1" class="ng-binding ng-scope">
    Mr Jules MARTIN
  </option>
  <!-- end ngRepeat: element in model.data -->
  <option ng-repeat="element in model.data" value="2" class="ng-binding ng-scope">
    Mme Christine GERMAN
  </option>
  <!-- end ngRepeat: element in model.data -->
  <option ng-repeat="element in model.data" value="3" class="ng-binding ng-scope">
    Mr Jules JACQUARD
  </option>
  <!-- end ngRepeat: element in model.data -->
  <option ng-repeat="element in model.data" value="4" class="ng-binding ng-scope">
    Melle Brigitte BISTROU
  </option>
  <!-- end ngRepeat: element in model.data -->
</select>
```

- en [1], on n'a que quatre éléments au lieu de huit ;
- en [2], ces quatre éléments sont dans un [select] mais celui-ci est caché (style='display : none');

```

    ...
    ▼<div class="dropdown-menu open" style="max-height: 268px; overflow: hidden; min-height: 92px;">
      ▼<ul class="dropdown-menu inner selectpicker" role="menu" style="max-height: 256px; overflow-y: auto; min-height: 80px;">
        ▼<li rel="0" class="selected">
          ▼<a tabindex="0" class="ng-binding ng-scope" style>
            <span class="text">
              Mr Jules MARTIN
            </span>
            <i class="glyphicon glyphicon-ok icon-ok check-mark"></i>
          </a>
        </li>
        ▼<li rel="1">
          ▼<a tabindex="0" class="ng-binding ng-scope" style>
            <span class="text">
              Mme Christine GERMAN
            </span>
            <i class="glyphicon glyphicon-ok icon-ok check-mark"></i>
          </a>
        </li>
        ▼<li rel="2">
          ▼<a tabindex="0" class="ng-binding ng-scope" style>
            <span class="text">
              Mr Jules JACQUARD
            </span>
            <i class="glyphicon glyphicon-ok icon-ok check-mark"></i>
          </a>
        </li>
        ▼<li rel="3">
          ▼<a tabindex="0" class="ng-binding ng-scope" style>
            <span class="text">
              Melle Brigitte BISTROU
            </span>
            <i class="glyphicon glyphicon-ok icon-ok check-mark"></i>
          </a>
        </li>
      </ul>
    </div>

```

3

- en [3], on retrouve les quatre clients dans une autre architecture HTML et c'est celle-ci que l'utilisateur voit lorsqu'il clique sur la liste déroulante ;

Enfin, les logs console disent la chose suivante :

```

1. [dao] init
2. [dao] getData[/getAllClients] success réponse : {"data":{"status":0,"data":
[{"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"}, {"id":2,"version":1,"titre":"Mme","nom":"GERMAN","prenom":"Christine"}, {"id":3,"version":1,"titre":"Mr","nom":"JACQUARD","prenom":"Jules"}, {"id":4,"version":1,"titre":"Melle","nom":"BISTROU","prenom":"Brigitte"}]}, "status":200, "config": {"method":"GET", "transformRequest": [null], "transformResponse": [null], "timeout":1000, "url":"http://localhost:8080/getAllClients", "headers":{"Accept":"application/json, text/plain, /*", "Authorization":"Basic YWRtaW46YWRtaW4="}}, "statusText":"OK"}
3. directive list2
4. [dao] getData[/getAllClients] success réponse : {"data":{"status":0,"data":
[{"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"}, {"id":2,"version":1,"titre":"Mme","nom":"GERMAN","prenom":"Christine"}, {"id":3,"version":1,"titre":"Mr","nom":"JACQUARD","prenom":"Jules"}, {"id":4,"version":1,"titre":"Melle","nom":"BISTROU","prenom":"Brigitte"}]}, "status":200, "config": {"method":"GET", "transformRequest": [null], "transformResponse": [null], "timeout":1000, "url":"http://localhost:8080/getAllClients", "headers":{"Accept":"application/json, text/plain, /*", "Authorization":"Basic YWRtaW46YWRtaW4="}}, "statusText":"OK"}

```

- ligne 1 : le service [dao] est instancié ;
- ligne 2 : le service [dao] obtient une première liste de clients ;
- ligne 3 : la directive [list2] est exécutée ;
- ligne 4 : le service [dao] obtient une seconde liste de clients ;

L'affichage de la ligne 2 vient du code suivant dans la directive :

```

1.   link: function (scope, element, attrs) {
2.     utils.debug('directive list2');
3.     ...
4.   }

```

Examinons le cycle de vie de la directive [list2] :

- entre les lignes 1 et 2, elle n'est pas activée alors que la vue a été affichée une première fois. C'est dû à son attribut [**ng-if="clients.show"**] dans la vue V :

```
<list2 model="clients" ng-if="clients.show"></list2>
```

- ligne 3 : après l'obtention de la première liste de médecins, [clients.show] passe à true et la directive est activée ;
- après l'obtention de la seconde liste de clients, on voit que le code de la directive [list2] n'est pas appelé. C'est pourquoi, on ne voit pas la seconde liste ;

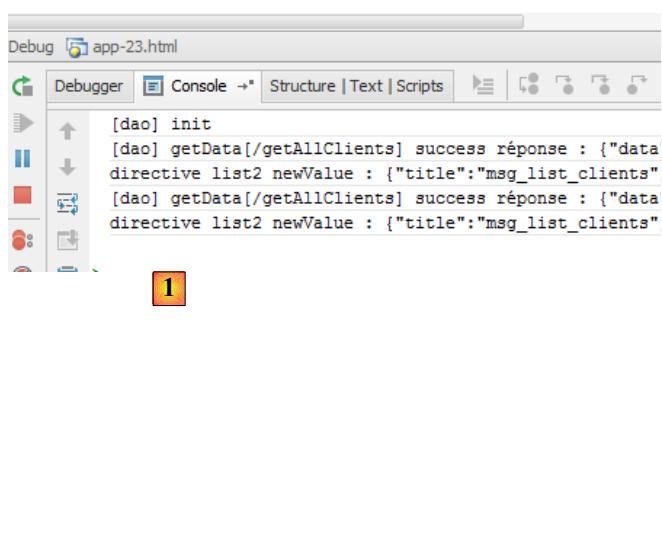
Pour résoudre ce problème, nous modifions la directive [list2] de la façon suivante :

```
1. angular.module("rdvmedecins")
2.   .directive("list2", ['utils', '$timeout', function (utils, $timeout) {
3.     // instance de la directive renournée
4.     return {
5.       // élément HTML
6.       restrict: "E",
7.       // url du fragment
8.       templateUrl: "list.html",
9.       // scope unique à chaque instance de la directive
10.      scope: true,
11.      // fonction lien avec le document
12.      link: function (scope, element, attrs) {
13.        // à chaque fois que attrs["model"] change, le modèle de la directive doit changer également
14.        scope.$watch(attrs["model"], function (newValue) {
15.          utils.debug("directive list2 newValue", newValue);
16.          // on met à jour le modèle de la directive
17.          scope.model = newValue;
18.          $timeout(function () {
19.            $('#' + scope.model.id).selectpicker('refresh');
20.          })
21.        });
22.      }
23.    }
24.  }]);

```

- ligne 14 : la fonction [scope.\$watch] permet d'observer une valeur du modèle. Sa syntaxe est [scope.\$watch('var'), f] où [var] est l'identifiant d'une variable du modèle et f la fonction à exécuter lorsque cette variable change de valeur. Ici, nous voulons observer la variable [clients]. Donc on doit écrire [scope.\$watch('clients')]. Comme on a attrs['model']='clients', on écrit [scope.\$watch(attrs["model"], function (newValue)] ;
- ligne 14 : le second paramètre de la fonction [scope.\$watch] est la fonction à exécuter lorsque la variable observée change de valeur. Le paramètre [newValue] est la nouvelle valeur de la variable, donc pour nous la nouvelle valeur de la variable [clients] du modèle ;
- ligne 17 : cette nouvelle valeur est affectée au champ [model] du modèle de la directive ;

Cette modification faite, les logs changent :



2

Ci-dessus, on voit qu'après avoir obtenu la seconde liste de clients, la directive [list2] est bien exécutée de nouveau, ce que confirme le résultat [2].

3.7.14 Exemple 14 : les directives [waiting] et [errors]

Revenons au code HTML de l'application précédente :

```
1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <!-- le message d'attente -->
5.   <div class="alert alert-warning" ng-show="waiting.visible">
6.   ...
7.   </div>
8.
9.   <!-- la liste d'erreurs -->
10.  <div class="alert alert-danger" ng-show="errors.show">
11.  ...
12.  </div>
13.
14.  <!-- le bouton -->
15.  <div class="alert alert-warning">
16.    <button class="btn btn-primary" ng-click="getClients()">{{clients.title|translate}}</button>
17.  </div>
18.
19.  <!-- la liste des clients -->
20.  <list2 model="clients" ng-if="clients.show"></list2>
21. </div>
```

- lignes 5-7 : le message d'attente ;
- lignes 10-12 : le message d'erreurs ;

Nous décidons de mettre les codes HTML de ces deux messages dans des directives.

3.7.14.1 Le nouveau code HTML

Le nouveau code HTML [app-24.html] est le suivant :

```
1. <div class="container">
2.   <h1>Rdvmedecins - v1</h1>
3.
4.   <!-- le message d'attente -->
5.   <waiting model="waiting"></waiting>
6.
7.   <!-- la liste d'erreurs -->
8.   <errors model="errors"></errors>
9.
10.  <!-- le bouton -->
11.  <div class="alert alert-warning">
12.    <button class="btn btn-primary" ng-click="getClients()">{{clients.title|translate}}</button>
13.  </div>
14.
15.  <!-- la liste des clients -->
16.  <list2 model="clients" ng-if="clients.show"></list2>
17. </div>
18. ...
19. <script type="text/javascript" src="rdvmedecins-12.js"></script>
20. <!-- directives -->
21. <script type="text/javascript" src="List2.js"></script>
22. <script type="text/javascript" src="errors.js"></script>
23. <script type="text/javascript" src="waiting.js"></script>
```

- ligne 5 : la directive pour le message d'attente ;
- ligne 8 : la directive pour le message d'erreurs ;
- ligne 19 : le nouveau fichier JS associé à l'application ;
- lignes 21-23 : les fichiers JS des trois directives ;

3.7.14.2 La directive [waiting]

Le code JS de la directive [waiting] est dans le fichier [waiting.js] suivant :

```
1. angular.module("rdvmedecins")
2.   .directive("waiting", ['utils', function (utils) {
3.     // instance de la directive retournée
4.     return {
5.       // élément HTML
6.       restrict: "E",
7.       // url du fragment
8.       templateUrl: "waiting.html",
9.       // scope unique à chaque instance de la directive
10.      scope: true,
11.      // fonction lien avec le document
12.      link: function (scope, element, attrs) {
13.        // à chaque fois que attr["model"] change, le modèle de la page doit changer également
14.        scope.$watch(attrs["model"], function (newValue) {
15.          utils.debug("[waiting] watch newValue", newValue);
16.          scope.model = newValue;
17.        });
18.      }
19.    }
20.  }]);
```

Ce code suit la même logique que celui de la directive [list2] déjà étudiée.

Ligne 8, on référence le fichier [waiting.html] suivant :

```
1. <div class="alert alert-warning" ng-show="model.show">
2.   <h1>{{ model.title.text | translate: model.title.values }}</h1>
3.   <button class="btn btn-primary pull-right" ng-click="model.cancel()">{{'cancel'|translate}}</button>
4.   
5. </h1>
6. </div>
```

Dans le code JS de l'application, le modèle [\$scope.waiting] de ce code HTML sera défini de la façon suivante :

```
// le msg d'attente
$scope.waiting = {title: {text: config.msgWaiting, values: {}}, show: false, cancel: cancel, time: 3000};
```

3.7.14.3 La directive [errors]

Le code JS de la directive [errors] est dans le fichier [errors.js] suivant :

```
1. angular.module("rdvmedecins")
2.   .directive("errors", ['utils', function (utils) {
3.     // instance de la directive retournée
4.     return {
5.       // élément HTML
6.       restrict: "E",
7.       // url du fragment
8.       templateUrl: "errors.html",
9.       // scope unique à chaque instance de la directive
10.      scope: true,
11.      // fonction lien avec le document
12.      link: function (scope, element, attrs) {
13.        // à chaque fois que attr["model"] change, le modèle de la page doit changer également
14.        scope.$watch(attrs["model"], function (newValue) {
15.          utils.debug("[errors] watch newValue", newValue);
16.          scope.model = newValue;
17.        });
18.      }
19.    }
20.  }]);
```

Ce code suit la même logique que celui de la directive [list2] déjà étudié.

Ligne 8, on référence le fichier [errors.html] suivant :

```

1. <div class="alert alert-danger" ng-show="model.show">
2.   {{model.title.text|translate:model.title.values}}
3.   <ul>
4.     <li ng-repeat="message in model.messages">{{message|translate}}</li>
5.   </ul>
6. </div>

```

Dans le code JS de l'application, le modèle [\$scope.errors] de ce code HTML sera défini de la façon suivante :

```

// il y a eu des erreurs pour obtenir la liste des clients
$scope.errors = { title: { text: config.getClientsErrors, values: {}}, messages: utils.getErrors(result), show: true, model: {}};

```

3.7.15 Exemple 15 : navigation

Jusqu'à maintenant, nous avons utilisé des applications à page unique. Nous abordons dans cet exemple, les applications à plusieurs pages et la navigation entre celles-ci.

3.7.15.1 Les vues V de l'application

Page 1 [2]

- Modèle global : [Modèle global]
- Modèle local : [Modèle local dans page 1]

Page 2 [4]

- Modèle global : [Modèle global]
- Modèle local : [Modèle local dans page 2]

- en [1], l'URL de la vue n° 1 ;
- en [2], son contenu ;
- en [3], on passe à la page 2 ;
- en [4], la vue n° 2 ;
- en [5], on passe à la page 3 ;

Page 3 [6]

- Modèle global : [Modèle global]
- Modèle local : [Modèle local dans page 3]

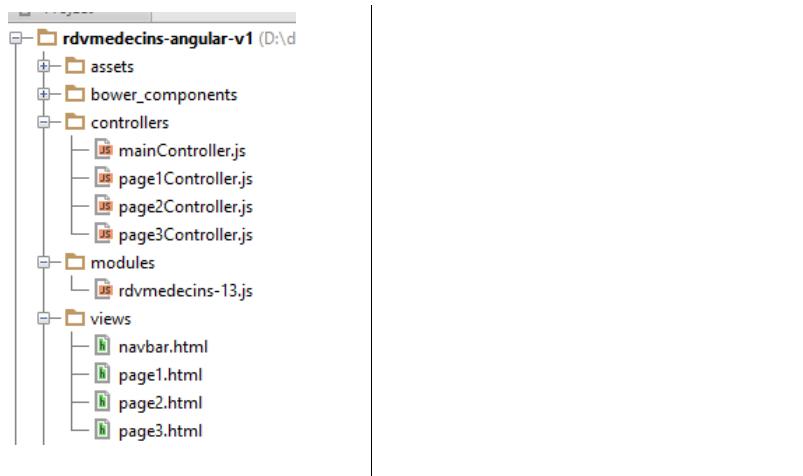
Page 1 [8]

- Modèle global : [Modèle global]
- Modèle local : [Modèle local dans page 1]

- en [6], la vue n° 3 ;
- en [7], on passe à la page 1 ;
- en [8], on est revenu à la vue n° 1 ;

3.7.15.2 Organisation du code

Nous commençons une nouvelle organisation du code :



- les vues de l'application seront placées dans le dossier [views] ;
- le module de l'application sera placé dans le dossier [modules] ;
- les contrôleurs de l'application seront placés dans le dossier [controllers] ;

De même, dans la version finale :

- les services seront placés dans le dossier [services] ;
- les directives seront placées dans le dossier [directives] ;

3.7.15.3 Le conteneur des vues

Les vues du dossier [views] seront affichées dans le conteneur suivant [app-25.html] :

```
1. <!DOCTYPE html>
2. <html ng-app="rdvmedecins">
3.   <head>
4.     ...
5.   </head>
6.   <body>
7.     <div class="container" ng-controller="mainCtrl">
8.       
9.       <ng-include src="'views/navbar.html'"></ng-include>
10.
11.      
12.      <ng-view></ng-view>
13.    </div>
14.
15. ...
16. <!-- le module -->
17. <script type="text/javascript" src="modules/rdvmedecins-13.js"></script>
18. <!-- les contrôleurs -->
19. <script type="text/javascript" src="controllers/mainController.js"></script>
20. <script type="text/javascript" src="controllers/page1Controller.js"></script>
21. <script type="text/javascript" src="controllers/page2Controller.js"></script>
22. <script type="text/javascript" src="controllers/page3Controller.js"></script>
23. </body>
24. </html>
```

- ligne 7 : le corps du conteneur est contrôlé par [mainCtrl] ;
- ligne 9 : la directive [ng-include] permet d'inclure un fichier HTML externe, ici une barre de navigation ;

- ligne 12 : les différentes vues affichées par le conteneur le sont à l'intérieur de la directive [ng-view]. Au final, on a un conteneur qui affiche :
 - toujours la même barre de navigation (ligne 9) ;
 - des vues différentes en ligne 12 ;
- lignes 16-22 : on importe les fichiers JS du module de l'application [[rdvmedecins-13.js](#)] et de ses contrôleurs ;

3.7.15.4 Le module de l'application

Le fichier [[rdvmedecins-13.js](#)] définit le module de l'application et le routage entre vues :

```

1. // -----
2. angular.module("rdvmedecins", [ 'ngRoute' ]);
3.
4. angular.module("rdvmedecins").config(["$routeProvider", function ($routeProvider) {
5. // -----
6.   $routeProvider.when("/page1",
7.   {
8.     templateUrl: "views/page1.html",
9.     controller: 'page1Ctrl'
10.    });
11.  $routeProvider.when("/page2",
12.   {
13.     templateUrl: "views/page2.html",
14.     controller: 'page2Ctrl'
15.   });
16.  $routeProvider.when("/page3",
17.   {
18.     templateUrl: "views/page3.html",
19.     controller: 'page3Ctrl'
20.   });
21.  $routeProvider.otherwise(
22.   {
23.     redirectTo: "/page1"
24.   });
25. }]);
```

- ligne 1 : on définit le module [rdvmedecins]. Il a une dépendance sur le module [ngRoute] fourni par la bibliothèque [[angular-route.min.js](#)]. C'est ce module qui permet le routage défini aux lignes 6-24 ;
- ligne 4 : définit la fonction [config] du module [rdvmedecins]. On rappelle que cette fonction est exécutée avant toute instanciation de service. C'est une fonction de configuration du module. Ici, c'est son routage qui est configuré. Ceci est fait au moyen de l'objet [\$routeProvider] fourni par le module [ngRoute] ;
- lignes 6-10 : définissent la vue à afficher lorsque l'utilisateur demande l'URL [/page1]. C'est un routage interne à l'application. L'URL est en fait [/rdvmedecins-angular-v1/app-21.html#/page1]. On voit que c'est toujours l'URL du conteneur [/rdvmedecins-angular-v1/app-21.html] qui est utilisée mais avec une information supplémentaire derrière un caractère #. C'est cette information supplémentaire que le routage Angular gère ;
- ligne 8 : indique le fragment HTML à insérer dans la directive [ng-view] du conteneur ;
- ligne 9 : indique le nom du contrôleur de ce fragment ;
- lignes 11-15 : définissent la vue à afficher lorsque l'utilisateur demande l'URL [/page2] ;
- lignes 16-20 : définissent la vue à afficher lorsque l'utilisateur demande l'URL [/page3] ;
- lignes 21-24 : définissent le routage à exercer lorsque l'URL demandée n'est pas l'une des trois précédentes (otherwise, ligne 21) ;
- ligne 23 : redirection vers l'URL [/page1], donc vers la vue définie aux lignes 6-10 ;

3.7.15.5 Le contrôleur du conteneur de vues

Nous avons vu que le conteneur de vues déclarait un contrôleur :

```
<div class="container" ng-controller="mainCtrl">
```

Le contrôleur [mainCtrl] est défini dans le fichier [mainController.js] :

```

1. // contrôleur
2. angular.module("rdvmedecins")
3.   .controller('mainCtrl', ['$scope', '$location',
4.     function ($scope, $location) {
```

```

5.      // modèles des pages
6.      $scope.page1 = {};
7.      $scope.page2 = {};
8.      $scope.page3 = {};
9.      // modèle global
10.     var main = $scope.main = {};
11.     main.text = "[Modèle global]";
12.
13.     // méthodes exposées à la vue
14.     main.showPage1 = function () {
15.       $location.path("/page1");
16.     };
17.     main.showPage2 = function () {
18.       $location.path("/page2");
19.     };
20.     main.showPage3 = function () {
21.       $location.path("/page3");
22.     }
23.   }
24. });

```

- ligne 3 : le contrôleur [mainCtrl] a besoin de l'objet [\$location] fourni par le module de routage [ngRoute]. Cet objet permet de changer de vue (lignes 16, 19, 22) ;

Revenons au code du conteneur :

```

1.   <div class="container" ng-controller="mainCtrl">
2.     <!-- la barre de navigation -->
3.     <ng-include src="'views/navbar.html'"></ng-include>
4.
5.     <!-- la vue courante -->
6.     <ng-view></ng-view>
7.   </div>

```

- le contrôleur [mainCtrl] construit le modèle de la zone 1-7 ;
- la vue incluse en ligne 6 a également un contrôleur. Par exemple la vue [page1] a le contrôleur [page1Ctrl]. Celui-ci construit le modèle de la zone affichée ligne 6. On a alors dans cette zone deux modèles :
 - le modèle construit par le contrôleur [mainCtrl] ;
 - le modèle construit par le contrôleur [page1Ctrl] ;

Il y a **héritage** des modèles. Dans la vue affichée ligne 6, les modèles des contrôleurs [mainCtrl] et [page1Ctrl] sont visibles tous les deux. Si deux variables de ces modèles portent le même nom, l'une va cacher l'autre. Pour éviter cette collision des noms, nous créons quatre modèles sous quatre noms :

page	contrôleur	modèle	ligne du code
conteneur	mainCtrl	main	11
page1	page1Ctrl	page1	7
page2	page2Ctrl	page2	8
page3	page3Ctrl	page3	9

- ligne 12 : définit un élément [text] dans le modèle [main] ;

Les lignes 7-11 ont une **conséquence très particulière** : elles définissent le [\$scope] du contrôleur [mainCtrl] et dans celui-ci, elles créent quatre variables [main, page1, page2, page3]. Ces quatre variables vont être utilisées comme modèles respectifs du conteneur et des trois vues qu'il va contenir tour à tour.

3.7.15.6 La barre de navigation

La barre de navigation est définie de la façon suivante dans le conteneur :

```

1.   <div class="container" ng-controller="mainCtrl">
2.     <!-- la barre de navigation -->
3.     <ng-include src="'views/navbar.html'"></ng-include>

```

```

4.
5.      <!-- la vue courante -->
6.      <ng-view></ng-view>
7.  </div>

```

La barre de navigation est définie ligne 3. Cela signifie qu'elle ne connaît que le modèle [main]. Son code est le suivant :

```

1.  <div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
2.    <div class="container">
3.      <div class="navbar-header">
4.        <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
5.          <span class="sr-only">Toggle navigation</span>
6.          <span class="icon-bar"></span>
7.          <span class="icon-bar"></span>
8.          <span class="icon-bar"></span>
9.        </button>
10.       <a class="navbar-brand" href="#">RdvMedecins</a>
11.     </div>
12.     <div class="collapse navbar-collapse">
13.       <ul class="nav navbar-nav">
14.         <li class="active">
15.           <a href="">
16.             <span ng-click="main.showPage1()">Page 1</span>
17.           </a>
18.         </li>
19.         <li class="active">
20.           <a href="">
21.             <span ng-click="main.showPage2()">Page 2</span>
22.           </a>
23.         </li>
24.         <li class="active">
25.           <a href="">
26.             <span ng-click="main.showPage3()">Page 3</span>
27.           </a>
28.         </li>
29.       </ul>
30.     </div>
31.   </div>
32. </div>

```

- aux lignes 16, 21, 26, ce sont des méthodes du modèle [main] qui sont utilisées ;
- ligne 16 : un clic sur le lien [Page1] va lancer l'exécution de la méthode `[$scope.main.showPage1]`. Celle-ci est définie dans le contrôleur [mainCtrl] de la façon suivante :

```

1.      // modèle global
2.      var main = $scope.main = {};
3.      main.text = "[Modèle global]";
4.
5.      // méthodes exposées à la vue
6.      main.showPage1 = function () {
7.        $location.path("/page1");
8.      };

```

- ligne 6 : du code qui précède, on voit que la méthode [main.showPage1] est en réalité la méthode `[$scope.main.showPage1]`. C'est donc bien celle-ci qui va s'exécuter ;
- ligne 7 : on change l'URL de l'application qui devient [/page1]. Revenons au routage qui a été défini dans le module principal :

```

1.      $routeProvider.when("/page1",
2.      {
3.        templateUrl: "views/page1.html",
4.        controller: 'page1Ctrl'
5.      });

```

on voit que le fragment [views/page1.html] va être inséré dans le conteneur et que son contrôleur est [page1Ctrl].

3.7.15.7 La vue [/page1] et son contrôleur

Le fragment [views/page1.html] est le suivant :

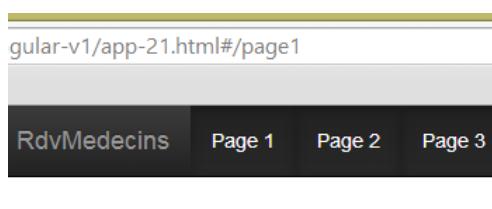
```
1. <h1>Page 1</h1>
2. <div class="alert alert-info">
3.   <ul>
4.     <li>Modèle global : {{main.text}}</li>
5.     <li>Modèle local : {{page1.text}}</li>
6.   </ul>
7. </div>
```

On se rappelle que dans la vue insérée dans le conteneur, le modèle [main] est visible. C'est ce qu'on veut vérifier ligne 4. Par ailleurs, le contrôleur [page1Ctrl] du fragment [views/page1.html] définit un modèle [page1]. C'est lui qui est utilisé ligne 5.

Le code du contrôleur [page1Ctrl] est le suivant :

```
1. angular.module("rdvmedecins")
2.   .controller('page1Ctrl', ['$scope',
3.     function ($scope) {
4.
5.       // modèle de la page 1
6.       var page1=$scope.page1;
7.       page1.text="[Modèle local dans page 1]";
8.   }]);
```

- ligne 2 : le [\$scope] injecté ici **n'est pas vide**. Puisque le contrôleur [page1Ctrl] contrôle une zone insérée dans un conteneur contrôlé par [mainCtrl], le [\$scope] de la ligne 2 contient les éléments du [\$scope] défini par le contrôleur [mainCtrl]. Il est important de le comprendre. Le [\$scope] défini par le contrôleur [mainCtrl] contient les éléments suivants [main, page1, page2, page3]. **Cela signifie qu'on a accès aux modèles de toutes les vues**. Ce n'est pas forcément désirable mais c'est le cas ici. Dans la version finale du client Angular, nous utiliserons cette particularité pour stocker dans le modèle [main] les informations qui doivent être partagées entre vues. On aura là, un concept analogue au concept de 'session' côté serveur ;
- ligne 6 : on récupère dans le [\$scope] le modèle [page1] de la page 1 et ensuite on travaille avec (ligne 7). On obtient alors l'affichage suivant :



Page 1

- Modèle global : [Modèle global]
- Modèle local : [Modèle local dans page 1]

Les vues [/page2] et [/page3] sont construits sur le même modèle que la vue [/page1] (voir les copies d'écran page 252).

3.7.15.8 Contrôle de la navigation

Nous souhaitons maintenant contrôler la navigation de la façon suivante [page1 --> page2 --> page3 --> page1]. Ainsi si l'utilisateur est sur la page 1 [/page1] et qu'il tape dans son navigateur l'URL [/page3] alors cette navigation ne doit pas être acceptée et on doit rester sur la page 1.

Pour obtenir ce résultat, nous modifions les contrôleurs des pages de la façon suivante :

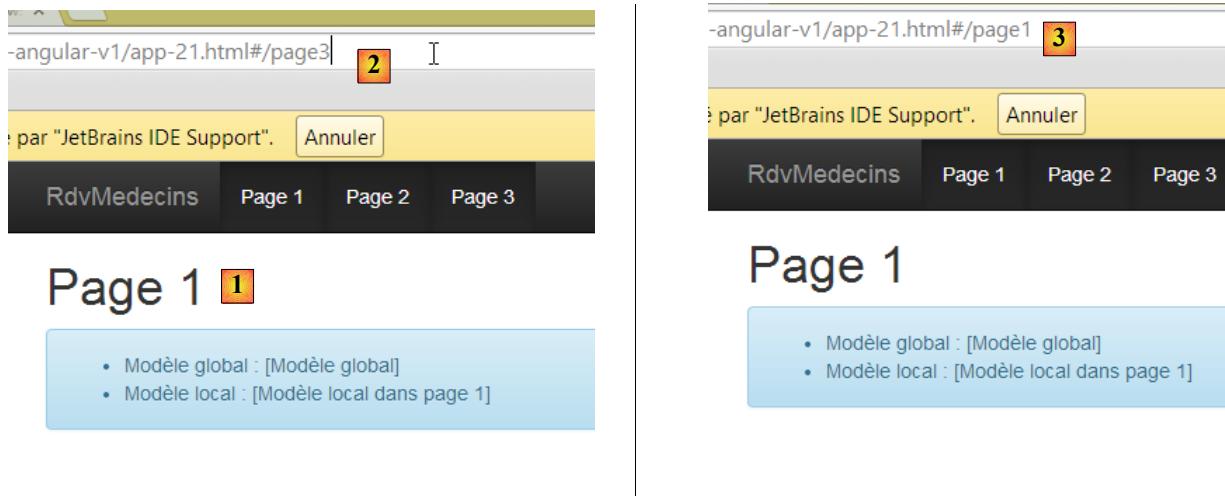
```

1. angular.module("rdvmedecins")
2.   .controller('page1Ctrl', ['$scope', '$location',
3.     function ($scope, $location) {
4.       // navigation autorisée ?
5.       var main = $scope.main;
6.       if (main.lastUrl && main.lastUrl != '/page3') {
7.         // on revient à la dernière URL
8.         $location.path(main.lastUrl);
9.         return;
10.      }
11.      // on mémorise l'URL de la page
12.      main.lastUrl = '/page1';
13.      // modèle de la page
14.      var page1 = $scope.page1;
15.      page1.text = "[Modèle local dans page 1]";
16.    }]);

```

- ligne 12 : lorsqu'une page sera affichée, on mémorisera son URL dans le modèle [main.lastUrl]. Nous utilisons ici le concept dont nous avons parlé précédemment : utiliser le modèle [main] pour stocker des informations partagées par toutes les vues. Ici, c'est la dernière URL consultée ;
- le code des lignes 4-12 est dupliqué et adapté aux trois vues. Ici on est dans la vue [/page1] ;
- ligne 5 : on récupère le modèle [main] ;
- ligne 6 : si le modèle [main.lastUrl] existe et s'il est différent de [/page3] alors la navigation est interdite (la dernière URL visitée existe et n'est pas /page3) ;
- ligne 8 : on revient alors sur la dernière URL visitée ;

Faisons un essai :



- en [1], on est sur la page 1 et on tape l'URL de la page 3 en [2] ;
- en [3], la navigation n'a pas eu lieu et on est revenu sur l'URL de la page 1 ;

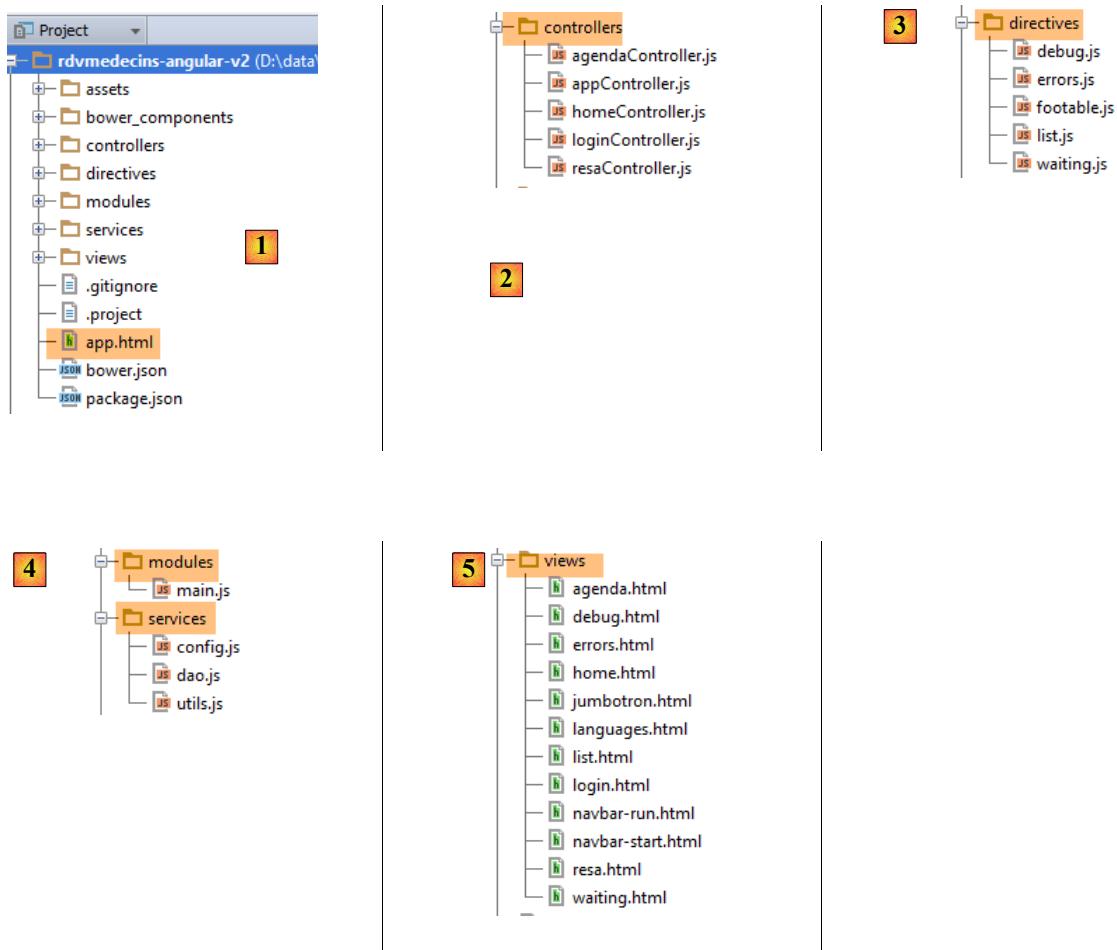
3.7.16 Conclusion

Nous avons balayé tous les cas d'utilisation que nous allons rencontrer dans la version finale du client Angular. Lorsque nous allons présenter celui-ci, nous commenterons davantage les fonctionnalités de l'application que ses détails d'implémentation. Pour ces derniers, nous nous contenterons de faire référence à l'exemple illustrant le cas d'utilisation alors étudié.

3.8 Le client final Angular

3.8.1 Structure du projet

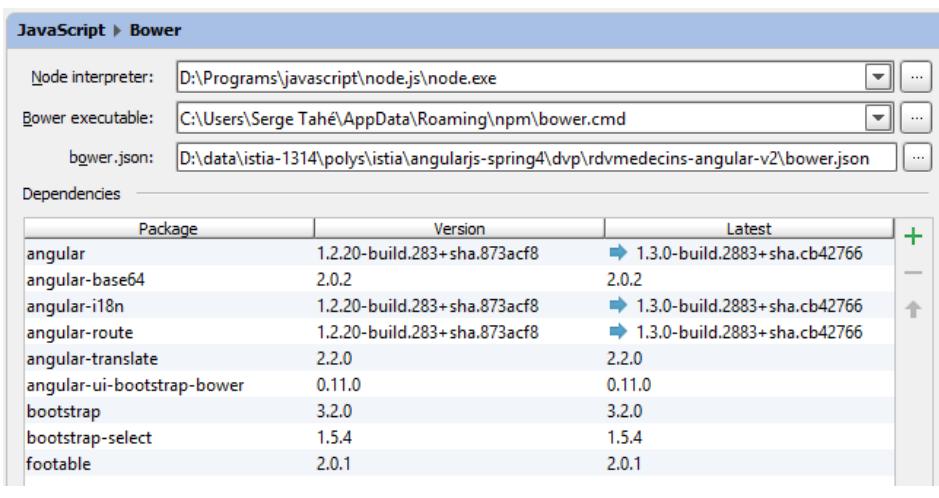
Le projet final a l'allure suivante :



- en [1], l'ensemble du projet. `[app.html]` est la page maître de l'application ;
- en [2], les contrôleurs ;
- en [3], les directives ;
- en [4], les services et le module Angular `[main.js]` de l'application ;
- en [5], les différentes vues qui viennent s'insérer dans la page maître `[app.html]` ;

3.8.2 Les dépendances du projet

Les dépendances du projet sont les suivantes :



Le rôle de ces différents éléments a été expliqué au paragraphe 3.4, page 140.

3.8.3 La page maître [app.html]

La page maître est la suivante :

```

1. <!DOCTYPE html>
2. <html ng-app="rdvmedecins">
3.   <head>
4.     <title>RdvMedecins</title>
5.     <!-- META -->
6.     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
7.     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8.     <meta name="description" content="Angular client for RdvMedecins">
9.     <meta name="author" content="Serge Tahé">
10.    <!-- le CSS -->
11.    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.min.css"/>
12.    <link href="bower_components/bootstrap/dist/css/bootstrap-theme.min.css" rel="stylesheet"/>
13.    <link href="bower_components/bootstrap-select/bootstrap-select.min.css" rel="stylesheet"/>
14.    <link href="assets/css/rdvmedecins.css" rel="stylesheet"/>
15.    <link href="assets/css/footable.core.min.css" rel="stylesheet"/>
16.  </head>
17.  <!-- contrôleur [appCtrl], modèle [app] -->
18.  <body ng-controller="appCtrl">
19.  <div class="container">
20.  ...
21.  </div>
22.  <!-- Bootstrap core JavaScript ===== -->
23.  <script type="text/javascript" src="bower_components/jquery/dist/jquery.min.js"></script>
24.  <script type="text/javascript" src="bower_components/bootstrap/dist/js/bootstrap.min.js"></script>
25.  <script type="text/javascript" src="bower_components/bootstrap-select/bootstrap-select.min.js"></script>
26.  <script src="bower_components/footable/js/footable.js" type="text/javascript"></script>
27.  <!-- angular js -->
28.  <script type="text/javascript" src="bower_components/angular/angular.min.js"></script>
29.  <script type="text/javascript" src="bower_components/angular-ui-bootstrap-bower/ui-bootstrap-tpls.min.js"></script>
30.  <script type="text/javascript" src="bower_components/angular-route/angular-route.min.js"></script>
31.  <script type="text/javascript" src="bower_components/angular-translate/angular-translate.min.js"></script>
32.  <script type="text/javascript" src="bower_components/angular-base64/angular-base64.min.js"></script>
33.  <!-- modules -->
34.  <script type="text/javascript" src="modules/main.js"></script>
35.  <!-- services -->
36.  <script type="text/javascript" src="services/config.js"></script>
37.  <script type="text/javascript" src="services/dao.js"></script>
38.  <script type="text/javascript" src="services/utils.js"></script>
39.  <!-- directives -->
40.  <script type="text/javascript" src="directives/waiting.js"></script>
41.  <script type="text/javascript" src="directives/errors.js"></script>
42.  <script type="text/javascript" src="directives/footable.js"></script>
43.  <script type="text/javascript" src="directives/debug.js"></script>
44.  <script type="text/javascript" src="directives/list.js"></script>
45.  <!-- controllers -->
```

```

46. <script type="text/javascript" src="controllers/appController.js"></script>
47. <script type="text/javascript" src="controllers/LoginController.js"></script>
48. <script type="text/javascript" src="controllers/homeController.js"></script>
49. <script type="text/javascript" src="controllers/agendaController.js"></script>
50. <script type="text/javascript" src="controllers/rescheduleController.js"></script>
51. </body>
52. </html>

```

- ligne 18 : on notera que [appCtrl] est le contrôleur de la page maître ;
- lignes 19-21 : le contenu de la page maître ;

Ce contenu est le suivant :

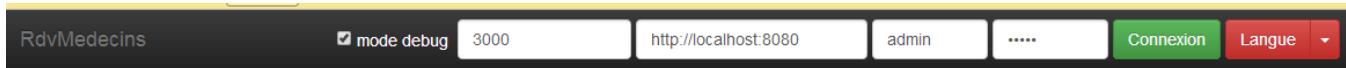
```

1. <div class="container">
2.   <!-- les barres de navigation -->
3.   <ng-include src="'views/navbar-start.html'" ng-show="app.navbarstart.show"></ng-include>
4.   <ng-include src="'views/navbar-run.html'" ng-show="app.navbarrun.show"></ng-include>
5.   <!-- le jumbotron -->
6.   <ng-include src="'views/jumbotron.html'"></ng-include>
7.   <!-- le titre de la page -->
8.   <div class="alert alert-info" ng-show="app.titre.show" translate="{{app.titre.text}}"
9.     translate-values="{{app.titre.model}}></div>
10.  <!-- les erreurs de la page -->
11.  <errors model="app.errors" ng-show="app.errors.show"></errors>
12.  <!-- le message d'attente -->
13.  <waiting model="app.waiting" ng-show="app.waiting.show"></waiting>
14.  <!-- la vue courante -->
15.  <ng-view></ng-view>
16.  <!-- debug -->
17.  <debug model="app" ng-show="app.debug.on"></debug>
18. </div>

```

Quelque soit la vue affichée, elle aura toujours les éléments suivants :

- lignes 3-4 : une barre de commande. Les deux barres des lignes 3 et 4 sont exclusives l'une de l'autre ;



- ligne 6 : un logo / texte de l'application :



Cabinet Médical Les Médecins Associés

- ligne 8 : un titre

Rendez - vous de Mme Marie PELISSIER le 2014-07-04

- ligne 11 : un message d'erreurs :

Les erreurs suivantes se sont produites lors de la validation du rendez-vous

- L'accès au serveur n'a pu être réalisé. Vérifiez sa disponibilité

- ligne 13 : un message d'attente :

Opération en cours. Patientez...

Annuler

- ligne 17 : une information de débogage :

```
Modèle : {"waitingTimeBeforeTask": "10000", "debug": { "on": true }, "titre": { "text": "agenda_titre", "model": { "titre": "Mme", "prenom": "Marie", "nom": "PELISSIER", "jour": "2014-07-04" }, "show": true }, "navbarnav": { "show": true }, "navbarstart": { "show": false }, "errors": { "show": false, "title": { "text": "post_resa_errors", "values": {} }, "messages": [ { "not_http_error": true } ], "view": { "url": "/agenda", "model": {}, "done": false }, "waiting": { "title": { "text": "msg_waiting", "values": {} }, "show": false }, "task": { "action": { "promise": {} }, "isFinished": false }, "serverUrl": "http://localhost:8080", "username": "admin", "password": "admin", "medecins": { "title": { "text": "list_medecins_title", "values": {} }, "data": [ { "id": 1, "version": 1, "titre": "Mme", "nom": "PELISSIER", "prenom": "Marie" }, { "id": 2, "version": 1, "titre": "Mr", "nom": "BROMARD", "prenom": "Jacques" }, { "id": 3, "version": 1, "titre": "Mr", "nom": "JANDOT", "prenom": "Philippe" }, { "id": 4, "version": 1, "titre": "Melle", "nom": "JACQUEMET", "prenom": "Justine" } ], "show": true, "id": "medecins" }, "clients": { "title": { "text": "list_clients_title", "values": {} }, "data": [ { "id": 1, "version": 1, "titre": "Mr", "nom": "MARTIN", "prenom": "Jules" }, { "id": 2, "version": 1, "titre": "Mme", "nom": "GERMAN", "prenom": "Christine" }, { "id": 3, "version": 1, "titre": "Mr", "nom": "JACQUARD", "prenom": "Jules" }, { "id": 4, "version": 1, "titre": "Melle", "nom": "BISTROU", "prenom": "Brigitte" } ], "show": true, "id": "clients" }, "menu": { "home": true }, "formattedJour": "2014-07-04", "agenda": { "medecin": { "id": 1, "version": 1, "titre": "Mme", "nom": "PELISSIER", "prenom": "Marie" }, "creneauxMedecin": [ { "rv": { "id": 19, "client": { "id": 1, "version": 1, "titre": "Mr", "nom": "MARTIN", "prenom": "Jules" }, "creneau": { "id": 1, "mDebut": 0, "mFin": 20, "hDebut": 8, "hFin": 8, "hDebut": 8 } }, "creneau": { "id": 1, "mDebut": 0, "mFin": 20, "hDebut": 8, "hFin": 8, "text": "08h00:08h20" } ], "rv": null, "creneau": { "id": 2, "mDebut": 20, "mFin": 40, "hDebut": 8, "hFin": 8, "text": "08h20:08h40" } ], "rv": null, "creneau": { "id": 3, "mDebut": 40, "mFin": 0, "hDebut": 9, "hFin": 9, "text": "08h40:09h00" } ], "rv": null, "creneau": { "id": 4, "mDebut": 0, "mFin": 20, "hDebut": 9, "hFin": 9, "text": "09h00:09h20" } ], "rv": null, "creneau": { "id": 5, "mDebut": 20, "mFin": 40, "hDebut": 9, "hFin": 9, "text": "09h20:09h40" } ], "rv": null, "creneau": { "id": 6, "mDebut": 40, "mFin": 0, "hDebut": 10, "hFin": 10, "text": "09h40:10h00" } ], "rv": null, "creneau": { "id": 7, "mDebut": 0, "mFin": 20, "hDebut": 10, "hFin": 10, "text": "10h00:10h20" } ], "rv": null, "creneau": { "id": 8, "mDebut": 20, "mFin": 40, "hDebut": 10, "hFin": 10, "text": "10h20:10h40" } ], "rv": null, "creneau": { "id": 9, "mDebut": 40, "mFin": 0, "hDebut": 11, "hFin": 11, "text": "10h40:11h00" } ], "rv": null, "creneau": { "id": 10, "mDebut": 0, "mFin": 20, "hDebut": 11, "hFin": 11, "text": "11h00:11h20" } ], "rv": null, "creneau": { "id": 11, "mDebut": 20, "mFin": 40, "hDebut": 11, "hFin": 11, "text": "11h20:11h40" } ], "rv": null, "creneau": { "id": 12, "mDebut": 40, "mFin": 0, "hDebut": 11, "hFin": 11, "text": "11h40:12h00" } ], "rv": null, "creneau": { "id": 13, "mDebut": 0, "mFin": 20, "hDebut": 14, "hFin": 14, "text": "14h00:14h20" } ], "rv": null, "creneau": { "id": 14, "mDebut": 20, "mFin": 40, "hDebut": 14, "hFin": 14, "text": "14h20:14h40" } ], "rv": null, "creneau": { "id": 15, "mDebut": 40, "mFin": 0, "hDebut": 14, "hFin": 14, "text": "14h40:15h00" } ], "rv": null, "creneau": { "id": 16, "mDebut": 0, "mFin": 20, "hDebut": 15, "hFin": 15, "text": "15h00:15h20" } ], "rv": null, "creneau": { "id": 17, "mDebut": 20, "mFin": 40, "hDebut": 15, "hFin": 15, "text": "15h20:15h40" } ], "rv": null, "creneau": { "id": 18, "mDebut": 40, "mFin": 0, "hDebut": 16, "hFin": 16, "text": "15h40:16h00" } ], "rv": null, "creneau": { "id": 19, "mDebut": 0, "mFin": 20, "hDebut": 16, "hFin": 16, "text": "16h00:16h20" } ], "rv": null, "creneau": { "id": 20, "mDebut": 20, "mFin": 40, "hDebut": 16, "hFin": 16, "text": "16h20:16h40" } ], "rv": null, "creneau": { "id": 21, "mDebut": 40, "mFin": 0, "hDebut": 17, "hFin": 16, "text": "16h40:17h00" } ], "rv": null, "creneau": { "id": 22, "mDebut": 0, "mFin": 20, "hDebut": 17, "hFin": 17, "text": "17h00:17h20" } ], "rv": null, "creneau": { "id": 23, "mDebut": 20, "mFin": 40, "hDebut": 17, "hFin": 17, "text": "17h20:17h40" } ], "rv": null, "creneau": { "id": 24, "mDebut": 40, "mFin": 0, "hDebut": 18, "hFin": 17, "text": "17h40:18h00" } ] }, "jour": "2014-07-04", "selectedCreneau": { "rv": null, "creneau": { "id": 1, "mDebut": 0, "mFin": 20, "hDebut": 8, "hFin": 8, "text": "08h00:08h20" } } }
```

Tous les éléments précédents sont contrôlés par une directive [ng-show / ng-hide] qui fait que s'ils sont bien présents, ils ne sont pas toujours visibles.

3.8.4 Les vues de l'application

Dans le code de la page maître, on a :

```
1. <div class="container">
2. ...
3.   <!-- la vue courante -->
4.   <ng-view></ng-view>
5. ...
6. </div>
```

La ligne 4 reçoit les différentes vues de l'application. Celles-ci sont définies dans le module [main.js] :

```
3 // ----- modules
4 angular.module("rdvmedecins", [...])
5 // ----- routage
6 .config(["$routeProvider", function ($routeProvider) {
7     $routeProvider.when("/", {
8         templateUrl: "views/login.html",
9         controller: 'loginCtrl'
10    });
11    $routeProvider.when("/home", {
12        templateUrl: "views/home.html",
13        controller: 'homeCtrl'
14    });
15    $routeProvider.when("/agenda", {
16        templateUrl: "views/agenda.html",
17        controller: 'agendaCtrl'
18    });
19    $routeProvider.when("/resa", {
20        templateUrl: "views/resat.html",
21        controller: 'resaCtrl'
22    });
23    $routeProvider.otherwise({
24        redirectTo: "/"
25    });
26 });
27 
```

Le rôle de la configuration des différentes routes a été expliqué au paragraphe 3.7.15.4, page 254.

La vue [login.html] est vide, c'est à dire qu'elle ne rajoute aucun élément à ceux déjà présents dans la page maître.

La vue [home.html] rajoute l'élément suivant à la page maître :

The screenshot shows a web application interface. At the top, there is a blue header bar with the title "Liste des médecins". Below the header, there is a dropdown menu set to "Mme Marie PELISSIER". Underneath the header, there is a section titled "Jour" which contains a date picker. The date picker displays the month of July 2014, with the days of the week labeled in French: dim., lun., mar., mer., jeu., ven., sam. The days of the month are numbered from 26 to 31. The day "31" is highlighted with a blue background, indicating it is the current date or selected date.

La vue [agenda.html] rajoute l'élément suivant à la page maître :

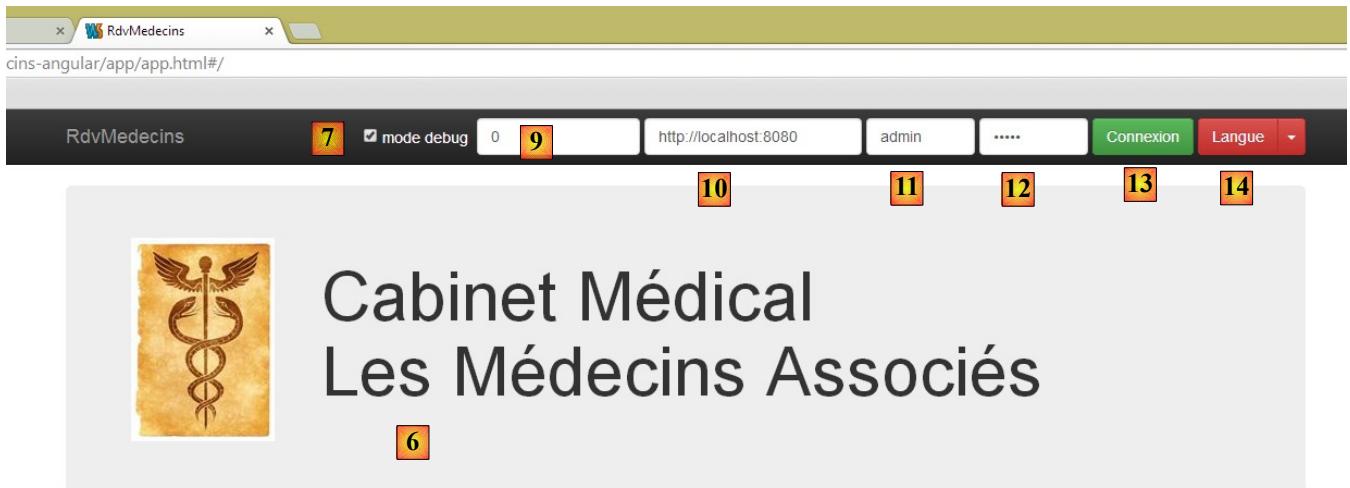
Créneau horaire	Client	Action
08h00:08h20	Mr Jules MARTIN	Supprimer
08h20:08h40		Réserver
08h40:09h00		Réserver
09h00:09h20		Réserver
09h20:09h40		Réserver
09h40:10h00		Réserver

La vue [resa.html] rajoute l'élément suivant à la page maître :



3.8.5 Fonctionnalités de l'application

Les vues du client Angular ont déjà été présentées au paragraphe 1.3.3, page 11. Pour faciliter la lecture de ce nouveau chapitre, nous les redonnons ici. La première vue est la suivante :



Authentifiez-vous pour accéder à l'application.

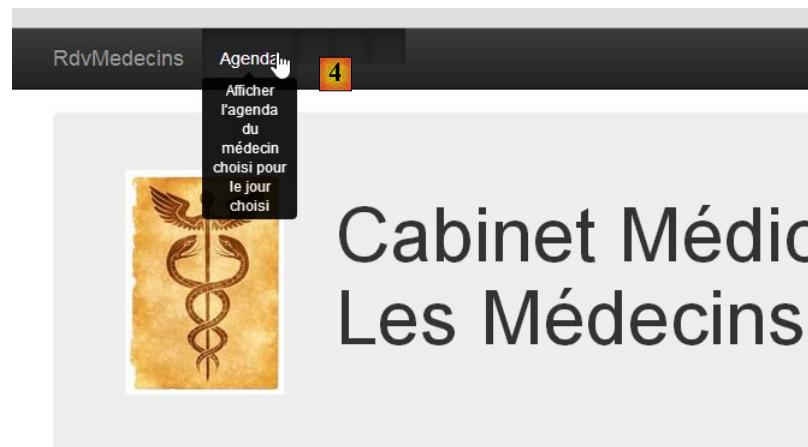
Pour connaître le rôle des différentes saisies, passez le curseur dessus pour avoir une aide.

```
Modèle : { "waitingTimeBeforeTask": 0, "delay": "0", "on": true, "titre": { "text": "identification", "show": true, "model": {} }, "navbarrun": { "show": false }, "navbarstart": { "show": true }, "errors": { "show": false }, "view": { "url": "identification", "model": {}, "done": false }, "waiting": { "text": "msg_waiting_init", "show": false }, "task": { "action": "promise", "promise": {} }, "isFinished": false }, "serverUrl": "http://localhost:8080", "username": "admin", "password": "admin" }
```

- en [6], la page d'entrée de l'application. Il s'agit d'une application de prise de rendez-vous pour des médecins ;
- en [7], une case à cocher qui permet d'être ou non en mode [debug]. Ce dernier se caractérise par la présence du cadre [8] qui affiche le modèle de la vue courante ;
- en [9], une durée d'attente artificielle en millisecondes. Elle vaut 0 par défaut (pas d'attente). Si N est la valeur de ce temps d'attente, toute action de l'utilisateur sera exécutée après un temps d'attente de N millisecondes. Cela permet de voir la gestion de l'attente mise en place par l'application ;
- en [10], l'URL du serveur Spring 4. Si on suit ce qui a précédé, c'est [http://localhost:8080];
- en [11] et [12], l'identifiant et le mot de passe de celui qui veut utiliser l'application. Il y a deux utilisateurs : **admin/admin** (login/password) avec un rôle (ADMIN) et **user/user** avec un rôle (USER). Seul le rôle ADMIN a le droit d'utiliser l'application. Le rôle USER n'est là que pour montrer ce que répond le serveur dans ce cas d'utilisation ;
- en [13], le bouton qui permet de se connecter au serveur ;
- en [14], la langue de l'application. Il y en a deux : le français par défaut et l'anglais.



- en [1], on se connecte ;



Médecin [2]

Jour [3]

Mme Marie PELISSIER

	dim.	lun.	mar.	mer.	jeu.	ven.	sam.
22	01	02	03	04	05	06	07
23	08	09	10	11	12	13	14
24	15	16	17	18	19	20	21
25	22	23	24	25	26	27	28
26	29	30	01	02	03	04	05
27	06	07	08	09	10	11	12

- une fois connecté, on peut choisir le médecin avec lequel on veut un rendez-vous [2] et le jour de celui-ci [3] ;
- on demande en [4] à voir l'agenda du médecin choisi pour le jour choisi ;



Cabinet Médical Les Médecins

Rendez - vous de Mme Marie PELISSIER le 2014-06-05

Créneau horaire	Client	Action
08h00:08h20		Réserver  
08h20:08h40		Réserver 
08h40:09h00		Réserver
09h00:09h20		Réserver

- une fois obtenu l'agenda du médecin, on peut réserver un créneau [5] ;



Cabinet Médical Les Médecins Ass

Rendez - vous de Mme Marie PELISSIER le 2014-06-05 à 08h00:08h20

Client 

Mr Jules MARTIN 

- en [6], on choisit le patient pour le rendez-vous et on valide ce choix en [7] ;



Cabinet Médical Les Médecins Associés

Rendez - vous de Mme Marie PELISSIER le 2014-06-05

Créneau horaire	Client	Action
08h00:08h20	Mr Jules MARTIN	Supprimer Supprimer le rendez-vous [7]
08h20:08h40		Réserver
08h40:09h00		Réserver
09h00:09h20		Réserver

Une fois le rendez-vous validé, on est ramené automatiquement à l'agenda où le nouveau rendez-vous est désormais inscrit. Ce rendez-vous pourra être ultérieurement supprimé [7].

Les principales fonctionnalités ont été décrites. Elles sont simples. Celles qui n'ont pas été décrites sont des fonctions de navigation pour revenir à une vue précédente. Terminons par la gestion de la langue :

RdvMedecins Agenda mode debug 0 Déconnexion Langue

French English [1]

Cabinet Médical Les Médecins Associés

Choisissez un médecin et un jour pour avoir l'agenda

Médecin

Mme Marie PELISSIER

Jour

juin 2014						
dim.	lun.	mar.	mer.	jeu.	ven.	sam.
22	01	02	03	04	05	06
23	08	09	10	11	12	13
24	15	16	17	18	19	20
25	22	23	24	25	26	27
26	29	30	01	02	03	04
27	06	07	08	09	10	11
						12

- en [1], on passe du français à l'anglais ;

The screenshot shows a web application interface. At the top, there is a navigation bar with links for "RdvMedecins", "Show the diary", "debug mode" (with a value of 0), "Logout", and "Language". The main title "The Associated Doctors" is displayed prominently. On the left side, there is a sidebar with a doctor's profile picture and the name "Mme Marie PELISSIER". Below the title, there is a button with the number "2" inside a red box. A message bar at the bottom says "Select a doctor and a date". The overall interface is in English.

- en [2], la vue est passée en anglais, y-compris le calendrier ;

3.8.6 Le module [main.js]

Le module [main.js] définit le module Angular qui va contrôler l'application :

```

3  // -----
4  angular.module("rdvmedecins", [
5    "ngRoute",
6    "pascalprecht.translate",
7    "base64",
8    | 'ngLocale',
9    | 'ui.bootstrap'
10   ])
11  // -----
12  + .config(["$routeProvider", function ($routeProvider) {...}])
13
14  // -----
15  + .config(function ($translateProvider) {...});
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182

```

- ligne 4 : le module s'appelle [rdvmedecins] ;
- ligne 5 : le module [ngRoute] est utilisé pour le routage des URL ;
- ligne 6 : le module [translate] est utilisé pour l'internationalisation des textes ;
- ligne 7 : le module [base64] est utilisé pour coder en Base64 la chaîne 'login:password' ;
- ligne 8 : le module [ngLocale] est utilisé pour internationaliser le calendrier ;
- ligne 9 : le module [ui.bootstrap] est utilisé pour le calendrier ;
- ligne 12 : la configuration des routes ;
- ligne 40 : l'internationalisation des messages ;

3.8.7 Le contrôleur de la page maître

Rappelons le code HTML de la page maître [app.html] :

```
1. <body ng-controller="appCtrl">
2. <div class="container">
3. ...
```

Ligne 1, tout le corps (body) de la page maître est contrôlé par le contrôleur [appCtrl]. De par sa position, cela en fait un contrôleur général et principal de l'application. Comme il a été expliqué au paragraphe 3.7.15, page 252, le modèle construit par ce contrôleur est hérité par toutes les vues qui viendront s'insérer dans la page maître.

Son code est le suivant :

```
1. angular.module("rdvmedecins")
2.   .controller("appCtrl", ['$scope', 'config', 'utils', '$location', '$locale',
3.     function ($scope, config, utils, $location, $locale) {
4.
5.       // debug
6.       utils.debug("[app] init");
7.
8.       // -----initialisation page
9.       // les modèles des # pages
10.      $scope.app = {waitingTimeBeforeTask: config.waitingTimeBeforeTask};
11.      $scope.login = {};
12.      $scope.home = {};
13.      $scope.agenda = {};
14.      $scope.resa = {};
15.      // modèle de la page courante
16.      var app = $scope.app;
17.      ...
18.
19.      // ----- méthodes
20.
21.      // annulation tâche courante
22.      app.cancel = function () {
23. ...
24.    };
25.
26.      // déconnexion
27.      app.deconnecter = function () {
28.        ...
29.      };
30.
31.      // ce code doit rester là car il référence la fonction [cancel] qui précède
32.      app.waiting = {title: {text: config.msgWaitingInit, values: {}}, cancel: app.cancel, show: true};
33.    }])
34. ;
```

Les lignes 10-14 définissent les cinq modèles qui sont utilisés dans l'application :

Modèle	Vue	Contrôleur
\$scope.app	app.html	appCtrl
\$scope.login	login.html	loginCtrl
\$scope.home	home.html	homeCtrl
\$scope.resa	resa.html	resaCtrl
\$scope.agenda	agenda.html	agendaCtrl

Ce qu'il est important de comprendre est que l'objet [\$scope] étant le modèle du contrôleur de la page maître, **est hérité par toutes les vues et contrôleurs**. Ainsi le contrôleur [loginCtrl] a accès aux éléments [\$scope.app, \$scope.login, \$scope.home, \$scope.resa, \$scope.agenda]. Dit autrement **un contrôleur a accès aux modèles des autres contrôleurs**. L'application étudiée évite soigneusement d'utiliser cette possibilité. Ainsi, par exemple, le contrôleur [loginCtrl] travaille avec deux modèles seulement :

- le sien [\$scope.login] ;
- et celui du contrôleur parent [\$scope.app] ;

Il en est de même pour tous les autres contrôleurs. Le modèle [\$scope.app] sera utilisé comme mémoire partagée entre les différents contrôleurs. Lorsqu'un contrôleur C1 devra transmettre de l'information au contrôleur C2, on procèdera ainsi :

Dans [C1] :

```
$scope.app.info=value ;
```

Dans [C2] :

```
var value=$scope.app.info ;
```

Dans les deux cas, `$scope` est hérité du contrôleur `[appCtrl]` et est donc identique (c'est un pointeur) dans [C1] et [C2]. L'objet `[$scope.app]` qui sert de mémoire partagée entre les contrôleurs sera souvent appelé *session* dans les commentaires, par mimétisme avec la session utilisée dans les applications web classiques qui désigne la mémoire partagée entre requêtes HTTP successives.

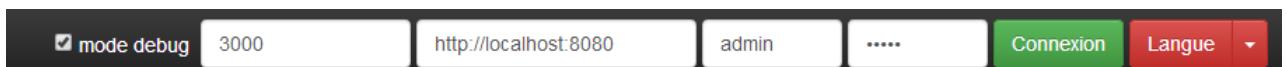
Revenons au code du contrôleur `[appCtrl]` :

```
1.      // les modèles des # pages
2.      $scope.app = {waitingTimeBeforeTask: config.waitingTimeBeforeTask};
3.      $scope.login = {};
4.      $scope.home = {};
5.      $scope.agenda = {};
6.      $scope.resa = {};
7.      // modèle de la page courante
8.      var app = $scope.app;
9.      // [app.debug] et [utils.verbose] doivent toujours être synchronisés
10.     app.debug = utils.verbose;
11.     app.debug.on = config.debug;
12.     // pas de titre de page pour l'instant
13.     app.titre = {show: false};
14.     // pas de barres de navigation
15.     app.navbarrun = {show: false};
16.     app.navbarstart = {show: false};
17.     // pas d'erreurs
18.     app.errors = {show: false};
19.     // locale par défaut
20.     angular.copy(config.locales['fr'], $locale);
21.     // la vue courante
22.     app.view = {url: undefined, model: {}, done: false};
23.     // la tâche courante
24.     app.task = app.view.model.task = {action: utils.waitForSomeTime(app.waitingTimeBeforeTask), isFinished: false};
```

- ligne 8 : `[$scope.app]` sera le modèle de la page maître. Ce sera aussi la mémoire partagée entre les différents contrôleurs. Plutôt que d'écrire partout `[$scope.app.champ=value]`, le pointeur `[$scope.app]` est affecté à la variable `[app]` et on écrira alors `[app.champ=value]`. Il faut simplement se souvenir que `[app]` est le modèle exposé à la page maître ;
- ligne 11 : `[app.debug.on]` est un booléen qui contrôle le mode *debug* de l'application. Par défaut il est à *true*. Sa valeur est liée à la case à cocher `[debug]` des barres de navigation ;
- ligne 15 : `[app.navbarrun.show]` contrôle l'affichage de la barre de navigation suivante :



- ligne 16 : `[app.navbarstart.show]` contrôle l'affichage de la barre de navigation suivante :



- ligne 18 : `[app.errors]` est le modèle du bandeau des erreurs ;

Les erreurs suivantes se sont produites lors du chargement de la liste des médecins

- Erreur d'authentification

- ligne 22 : [app.view] contiendra des informations sur la vue courante, celle qui est actuellement affichée par la balise [ng-view] de la page maître. Nous y noterons les informations suivantes :
 - [url] : l'URL de la vue courante, par exemple [/agenda] ;
 - [model] : le modèle de la vue courante, par exemple [\$scope.agenda] ;
 - [done] : à vrai, indique que la vue courante a terminé son travail et qu'on est en train de passer à une autre vue ;

Ces informations servent au contrôle de la navigation.

- ligne 24 : lance une tâche asynchrone, une attente simulée. La tâche asynchrone est référencée par deux pointeurs [app.view.model.task.action] et [app.task] ;

Deux méthodes ont été factorisées dans le contrôleur [appCtrl] :

```
1.      // annulation tâche courante
2.      app.cancel = function () {
3.      ...
4.      };
5.
6.      // déconnexion
7.      app.deconnecter = function () {
8.      ...
9.      };
```

- ligne 2 : la fonction [app.cancel] sert à annuler la tâche courante pour laquelle un message d'attente est actuellement affiché. Toutes les vues offrent ce message et donc l'annulation de la tâche se fera ici ;
- ligne 7 : la fonction [app.deconnecter] ramène l'utilisateur à la page d'authentification. Toutes les vues, sauf la vue [/login] offrent cette possibilité ;

La fonction [app.deconnecter] est la suivante :

```
1.      // déconnexion
2.      app.deconnecter = function () {
3.          // on revient à la page de login
4.          $location.path(config.urlLogin);
5.      };
```

- ligne 4 : on revient à la page de login d'URL [/login] ;

3.8.8 Gestion de la tâche asynchrone

Dans notre application, à un moment donné, une seule tâche asynchrone sera en cours d'exécution. Il est possible d'en avoir plusieurs. Par exemple, au démarrage de l'application, celle-ci demande au service web la liste des médecins et puis celle des clients avec deux requêtes HTTP successives. On pourrait faire la même chose avec deux requêtes HTTP simultanées. Angular offre les outils pour cette gestion. Ici, nous n'avons pas fait ce choix.

La tâche en cours d'exécution est annulée avec le code suivant dans le contrôleur [appCtrl] :

```
1.      // annulation tâche courante
2.      app.cancel = function () {
3.          utils.debug("[app] cancel task");
4.          // on annule la tâche asynchrone de la vue courante
5.          var task = app.view.model.task;
6.          task.isFinished = true;
7.          task.action.reject();
8.
9.          ...
10.     };
```

- ligne 5 : la tâche est cherchée dans [app.view.model.task]. Aussi, tous les contrôleurs feront en sorte que leurs tâches asynchrones soient référencées par cet objet ;

- ligne 6 : pour indiquer que la tâche est finie ;
- ligne 7 : pour terminer la tâche avec un échec. Cette notation est **différente** de celle utilisée dans les exemples Angular étudiés :
 - dans les exemples, l'objet [task] était un objet [\$q.defer()] qu'on pouvait terminer ;
 - dans la version finale, l'objet [task] est un objet avec les champs [action, isFinished] où [action] est l'objet [\$q.defer()] qu'on peut terminer et [isFinished] un booléen qui indique que l'action est terminée ;

Examinons le cycle de vie de l'objet [task] sur un exemple. Au démarrage, après le contrôleur [appCtrl], c'est le contrôleur [loginCtrl] qui prend la main pour afficher la vue [views/login.html]. Son code d'initialisation est le suivant :

```

1. // on récupère le modèle parent
2. var login = $scope.login;
3. var app = $scope.app;
4. // vue courante
5. app.view = {url: config.urlLogin, model: login, done: false};

```

Ligne 5, on a [model=login]. Ceci signifie que lorsqu'on modifie l'objet [login], on modifie l'objet [app.view.model] donc [\$scope.app.view.model]. Lorsque dans le contrôleur [loginCtrl], on veut faire une attente simulée, on écrit :

```
// attente simulée
var task = login.task = {action: utils.waitForSomeTime(app.waitingTimeBeforeTask), isFinished: false};
```

En ajoutant le champ [task] à l'objet [login], c'est donc à l'objet [\$scope.app.view.model] qu'il a été ajouté. Si l'utilisateur annule l'attente, le code dans [appCtrl.cancel] :

```

1. // modèle de la page courante
2. var app = $scope.app;
3. ...
4. var task = app.view.model.task;
5. task.isFinished = true;
6. task.action.reject();

```

va bien terminer l'attente simulée (lignes 4-6).

3.8.9 Contrôle de la navigation

Les règles de navigation utilisées dans l'application sont les suivantes :

URL cible	URL précédente	Navigation autorisée
/login	quelconque	oui
/home	/login	oui si le contrôleur [loginCtrl] a indiqué qu'il avait fini son travail
	/home	oui
	/agenda	oui
/agenda	/home	oui si le contrôleur [homeCtrl] a indiqué qu'il avait fini son travail
	/resa	oui
	/agenda	oui
/resa	/agenda	oui si le contrôleur [agendaCtrl] a indiqué qu'il avait fini son travail
	/resa	oui

Cela est implémenté avec le code suivant :

Pour [agendaCtrl] :

```

4  angular.module("rdvmedecins")
5    .controller('agendaCtrl', ['$scope', 'config', '$filter', '$translate', '$locale', '$location', 'utils', 'dao',
6      function ($scope, config, $filter, $translate, $locale, $location, utils, dao) {
7
8        // log
9        utils.debug("[agendaCtrl] init");
10       // contrôle de navigation
11       var lastView = $scope.app.view;
12       utils.debug("[agendaCtrl] lastView", lastView);
13       var navigationAllowed = lastView.url == config.urlAgenda
14       || lastView.url == config.urlResa
15       || (lastView.url == config.urlHome && lastView.done);
16       if (!navigationAllowed) {
17         // on ne bouge pas
18         $location.path(lastView.url);
19         return;
20     }
21     // ----- modèle de la page
22     // on récupère le modèle parent
23     var agenda = $scope.agenda;
24     var app = $scope.app;
25     // vue courante
26     app.view = {url: config.urlAgenda, model: agenda, done: false};
27     // les barres de navigation

```

- lignes 11-20 : implémentation de la règle de navigation ;
- ligne 26 : nouvelle vue courante ;

Pour [resaCtrl] :

```

4  angular.module("rdvmedecins")
5    .controller('resaCtrl', ['$scope', 'config', '$filter', '$translate', '$locale', '$location', 'utils', 'dao', {
6      function ($scope, config, $filter, $translate, $locale, $location, utils, dao) {
7
8        // log
9        utils.debug("[resaCtrl] init");
10       // contrôle de navigation
11       var lastView = $scope.app.view;
12       utils.debug("[resaCtrl] lastView", lastView);
13       var navigationAllowed = lastView.url == config.urlResa
14       || (lastView.url == config.urlAgenda && lastView.done);
15       if (!navigationAllowed) {
16         // on ne bouge pas
17         $location.path(lastView.url);
18         return;
19     }
20
21     // ----- initialisation vue
22     // on récupère le modèle parent
23     var resa = $scope.resa;
24     var app = $scope.app;
25     // vue courante
26     app.view = {url: config.urlResa, model: resa, done: false};
27     // ----- modèle de la page

```

- lignes 12-20 : implémentation de la règle de navigation ;
- ligne 27 : nouvelle vue courante ;

Pour [loginCtrl] :

```

4  angular.module("rdvmedecins")
5    .controller('loginCtrl', ['$scope', 'config', 'dao', '$location', '$filter', 'utils', '$translate', '$locale', {
6      function ($scope, config, dao, $location, $filter, utils, $translate, $locale) {
7
8        // log
9        utils.debug("[loginCtrl] init");
10
11       // ----- initialisation
12       // on récupère le modèle parent
13       var login = $scope.login;
14       var app = $scope.app;
15       // vue courante
16       app.view = {url: config.urlLogin, model: login, done: false};
17
18       // on attend la fin du chargement de l'application
19       app.task.action.promise.then(function () {
20         // fin attente

```

- il n'y a ici aucun contrôle de navigation puisque la règle dit qu'on peut venir à l'URL [/login] de n'importe où. Donc si l'utilisateur tape cette URL dans son navigateur, cela marchera quelque soit la vue courante du moment ;
- ligne 16 : la nouvelle vue courante ;

Le code pour le contrôleur [homeCtrl] est le suivant :

```
1 angular.module("rdvmedecins")
2   .controller('homeCtrl', ['$scope', 'config', '$filter', '$translate', '$locale', '$location', 'utils', 'dao'],
3     function ($scope, config, $filter, $translate, $locale, $location, utils, dao) {
4       // log
5       utils.debug("[homeCtrl] init");
6
7       // contrôle de navigation
8       var lastView = $scope.app.view;
9       utils.debug("[homeCtrl] lastView", lastView);
10      var navigationAllowed = lastView.url == config.urlHome
11        || lastView.url == config.urlAgenda
12        || (lastView.url == config.urlLogin && lastView.done);
13
14      if (!navigationAllowed) {
15        // on ne bouge pas
16        $location.path(lastView.url);
17        return;
18      }
19
20      // ----- initialisation du modèle de la page
21      // on récupère le modèle parent
22      var home = $scope.home;
23      var app = $scope.app;
24      // vue courante
25      app.view = {url: config.urlHome, model: home, done: false};
26      // préparation du modèle
27      app.navbarstart.show = false;
```

- lignes 8-18 : implémentation de la règle de navigation ;
- ligne 25 : nouvelle vue courante ;

Enfin pour une règle telle que :

/agenda | /home | oui si le contrôleur [homeCtrl] a indiqué qu'il avait fini son travail

voici un exemple de code qui fait passer de l'URL [/home] à l'URL [/agenda] :

```
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
```

```
...  
// on analyse le résultat  
promise.then(function (result) {  
  // fin d'attente  
  app.waiting.show = false;  
  // erreur ?  
  if (result.err == 0) {  
    // on met les données acquises dans la session  
    app.agenda = result.data;  
    // on passe à la vue suivante  
    app.view.done = true;  
    $location.path(config.urlAgenda);  
  } else {  
    // il y a eu des erreurs pour obtenir l'agenda  
    app.errors = {  
      title: {text: config.getAgendaErrors, values: {}},  
      messages: utils.getErrors(result, $filter),  
      show: true};  
  }  
});
```

Ci-dessus, on est dans la méthode [afficherAgenda] du contrôleur [homeCtrl]. L'utilisateur a demandé l'agenda d'un médecin.

- ligne 107 : la promesse de la tâche HTTP ;
- ligne 109 : la variable [app] a été initialisée avec [\$scope.app]. Ce dernier objet est, nous l'avons vu, utilisé comme modèle de la vue [app.html]. Ce modèle [\$scope.app] est également utilisé pour stocker l'information qui doit être partagée entre les vues ;
- ligne 111 : on analyse le code d'erreur renvoyé par la tâche ;

- ligne 113 : le résultat [result.data] est mis dans le modèle [app] ;
- ligne 116 : le contrôleur [homeCtrl] va passer la main au contrôleur [agendaCtrl]. Il lui indique qu'il a terminé son travail avec le code de la ligne 115. Ce code va être exploité par le contrôleur [agendaCtrl] de la façon suivante :

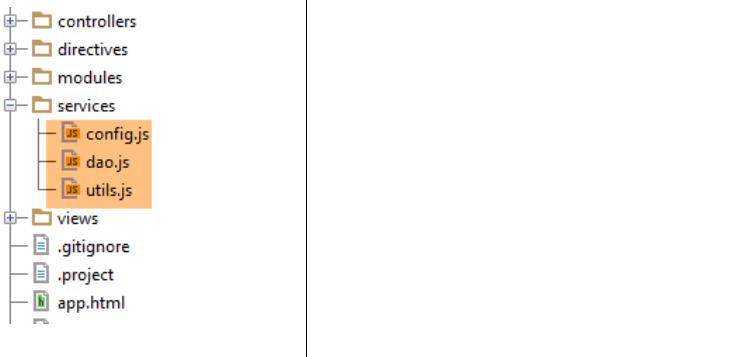
```

4   angular.module("rdvmedecins")
5     .controller('agendaCtrl', ['$scope', 'config', '$filter', '$translate',
6       function ($scope, config, $filter, $translate, $location,
7         $log, $utils) {
8           // log
9           $utils.debug("[agendaCtrl] init");
10          // contrôle de navigation
11          var lastView = $scope.app.view;
12          $utils.debug("[agendaCtrl] lastView", lastView);
13          var navigationAllowed = lastView.url == config.urlAgenda
14            || lastView.url == config.urlRensa
15            || (lastView.url == config.urlHome && lastView.done);
16          if (!navigationAllowed) {
17            // on ne bouge pas
18            $location.path(lastView.url);
19          }
20        }

```

- ligne 11 : l'objet [\$scope.app.view] est récupéré ;
- ligne 15 : exploitation du champ [\$scope.app.view.done] initialisé par [homeCtrl] ;

3.8.10 Les services



Les services [config, utils, dao] sont ceux déjà décrits lors de la présentation d'Angular :

- le service [config] a été introduit au paragraphe 3.7.4, page 173 ;
- le service [utils] a été introduit au paragraphe 3.7.5, page 177 ;
- le service [dao] a été introduit au paragraphe 3.7.6, page 184 ;

Pour mémoire, on rappelle la structure de ces services :

Service [config]

<pre> 4 angular.module("rdvmedecins") 5 .factory('config', function () { ... }) 248 ; 249 </pre>	<p>1</p>	<pre> 4 angular.module("rdvmedecins") 5 .factory('config', function () { 6 return { 7 // messages à internationaliser 8 msgWaitingInit: "msg_waiting_init", 9 msgWaiting: "msg_waiting", 10 loadingError: 'loading_error', 11 canceledOperation: 'canceled_operation', 12 getMedecinsErrors: 'get_medecins_errors', 13 getClientsErrors: 'get_clients_errors', </pre>	<p>2</p>
---	-----------------	---	-----------------

- en [1] : on voit que le code fait environ 250 lignes. L'essentiel de ce code est l'externalisation des clés des messages internatiomalisés [2]. On évite de mettre ces clés en dur dans le code ;

Service [utils]

```
4 angular.module("rdvmedecins")
5   .factory('utils', ['$config', '$timeout', '$q', function ($config, $timeout, $q) {
6
7     // mode debug
8     var verbose = {};
9     // attente
10    var waitForSomeTime = function (milliseconds) {...};
11
12    // affichage de la représentation Json d'un objet
13    function debug(message, data) {...}
14
15    // analyse des erreurs dans la réponse du serveur JSON
16    function getErrors(data) {...}
17
18    // mise en forme du libellé d'un créneau horaire
19    function getTextForCreneau(creneau) {...}
20
21    // mise en forme du libellé d'un créneau horaire - 2
22    function getTextFor(number) {...}
23
24    // instance du service
25    return {
26      waitForSomeTime: waitForSomeTime,
27      debug: debug,
28      getErrors: getErrors,
29      getTextForCreneau: getTextForCreneau,
30      verbose: verbose
31    };
32  }]);
33
```

- ligne 8 : nous n'avions pas encore rencontré la variable [verbose]. Elle contrôle la fonction [debug] de la façon suivante :

```
20 // affichage de la représentation Json d'un objet
21 function debug(message, data) {
22   if (verbose.on) {
23     var text = data ? message + " : " + angular.toJson(data) : message;
24     console.log(text);
25   }
26 }
```

- lignes 22-25 : la fonction [utils.debug] ne fait rien si [verbose.on] est évalué à *false*. Cette variable est liée à une variable du contrôleur [appCtrl] :

```
18 // modèle de la page courante
19 var app = $scope.app;
20 // [app.debug] et [utils.verbose] doivent toujours être synchronisés
21 app.debug = utils.verbose;
22 app.debug.on = config.debug;
23 // pas de titre de page pour l'instant
24 app.titre = {show: false};
25 // pas de barres de navigation
```

- ligne 21 : [app.debug] prend la valeur du pointeur [utils.verbose]. Donc toute modification faite sur [app.debug] sera faite également sur [utils.verbose] ;
- ligne 22 : la valeur initiale de [app.debug.on] est prise dans le fichier de configuration. Par défaut, c'est la valeur *true*. Cette valeur peut changer dans le temps. L'utilisateur a en effet la possibilité de la changer dans les barres de navigation :

```

1 <div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
2   <div class="container">
3     <div class="navbar-header" ...>
4     <div class="collapse navbar-collapse">
5       <ul class="nav navbar-nav" ...>
6         <!-- boutons de droite -->
7         <form class="navbar-form navbar-right" role="form">
8           <div class="checkbox" style="...">
9             <label>
10               <input type="checkbox" ng-model="app.debug.on" checked="checked" />
11               <span style="..." tooltip="{{'tooltip_debug'|translate}}" tooltip-placement="bottom">
12                 {{'verbose'|translate}}</span>
13             </label>
14           </div>
15           <div class="form-group" ...>
16             <!-- déconnexion -->
17             <button type="button" class="btn btn-success" translate="logout" ...>
18               <!-- langues -->
19               <ng-include src="'views/languages.html'"></ng-include>
20             </button>
21           </div>
22         </form>
23       </div>
24     </div>
25   </div>
26 </div>

```

- ligne 45 : une case à cocher (type=checkbox) permet de changer la valeur de [app.debug.on] (attribut ng-model) ;

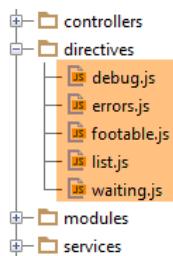
Service [dao]

```

4 angular.module("rdvmedecins")
5   .factory('dao', ['$http', '$q', 'config', '$base64', 'utils',
6     function ($http, $q, config, $base64, utils) {
7
8     // log
9     utils.debug("[dao] init");
10
11    // liste de données
12    function getData(serverUrl, username, password, urlAction, info) {...}
13
14    // ----- instance du service [dao]
15    return {
16      getData: getData
17    }
18  });

```

3.8.11 Les directives



Les directives [errors, footable, list, waiting] sont celles déjà décrites lors de la présentation d'Angular :

- la directive [footable] a été introduite page 224;
- la directive [list] a été introduite au paragraphe 3.7.12, page 241 ;
- les directives [errors] et [waiting] ont été introduites au paragraphe 3.7.14, page 250 ;

Nous n'avions pas rencontré la directive [debug]. C'est la suivante :

```

4   angular.module("rdvmedecins")
5     .directive("debug", ['$utils', function (_utils) {
6       // instance de la directive renournée
7       return {
8         // élément HTML
9         restrict: "E",
10        // url du fragment
11        templateUrl: "views/debug.html",
12        // scope unique à chaque instance de la directive
13        scope: true,]
14      // fonction lien avec le document
15      link: function (scope, element, attrs) {
16        // à chaque fois que attr["model"] change, le modèle de la page doit changer également
17        scope.$watch(attrs["model"], function (newValue) {
18          _utils.debug("[debug] watch newValue", newValue);
19          scope.model = newValue;
20        });
21      }
22    }
23  ]);

```

Le fichier [debug.html] référencé ligne 11 est le suivant :

```

1   <div class="alert alert-warning" style="margin-top: 20px">
2     Modèle : {{model|json}}
3   </div>

```

- ligne 2 : la directive [debug] affiche son modèle au format JSON dans un bandeau Bootstrap (ligne 1) ;

Cette directive n'est utilisée que dans la page maître [app.html] :

```

19  <div class="container">
20    <!-- les barres de navigation -->
21    <ng-include src="'views/navbar-start.html'" ng-show="app.navbarstart.show"></ng-include>
22    <ng-include src="'views/navbar-run.html'" ng-show="app.navbarrun.show"></ng-include>
23    <!-- le jumbotron -->
24    <ng-include src="'views/jumbotron.html'"></ng-include>
25    <!-- le titre de la page -->
26    <div class="alert alert-info" ng-show="app.titre.show" translate="{{app.titre.text}}"
27      translate-values="{{app.titre.model}}></div>
28    <!-- les erreurs de la page -->
29    <errors model="app.errors" ng-show="app.errors.show"></errors>
30    <!-- le message d'attente -->
31    <waiting model="app.waiting" ng-show="app.waiting.show"></waiting>
32    <!-- la vue courante -->
33    <ng-view></ng-view>
34    <!-- debug -->
35    <debug model="app" ng-show="app.debug.on"></debug>
36  </div>

```

- la directive [debug] est utilisée ligne 35. Elle affiche donc la forme JSON du modèle [\$scope.app] lorsqu'on est en mode debug (attribut ng-show). Cela donne des choses comme celle-ci :

```

Modèle : { "waitingTimeBeforeTask": "0", "debug": { "on": true }, "titre": { "text": "agenda_titre", "model": { "titre": "Mme", "prenom": "Marie", "nom": "PELISSIER", "jour": "2014-07-06" }, "show": true }, "navbarrun": { "show": true }, "navbarstart": { "show": false }, "errors": { "show": false }, "view": { "url": "/agenda", "model": { "task": { "action": { "promise": {} } }, "isFinished": false } }, "done": false }, "task": { "action": { "promise": {} }, "isFinished": false }, "waiting": { "title": { "text": "msg_waiting", "values": {} }, "show": false }, "serverUrl": "http://localhost:8080", "username": "admin", "password": "admin", "medecins": { "title": { "text": "list_medecins_title", "values": {} }, "data": [ { "id": 1, "version": 1, "titre": "Mme", "nom": "PELISSIER", "prenom": "Marie" }, { "id": 2, "version": 1, "titre": "Mr", "nom": "BROMARD", "prenom": "Jacques" }, { "id": 3, "version": 1, "titre": "Mr", "nom": "JANDOT", "prenom": "Philippe" }, { "id": 4, "version": 1, "titre": "Melle", "nom": "JACQUEMET", "prenom": "Justine" } ], "clients": { "title": { "text": "clients" } }, "menu": { "home": true }, "formattedJour": "2014-07-06", "agenda": { "medecin": { "id": 1, "version": 1, "titre": "Mme", "nom": "PELISSIER", "prenom": "Marie" }, "creneauxMedecin": { "rv": null, "creneau": { "id": 1, "mDebut": 0, "mFin": 20, "hFin": 8, "hDebut": 8, "text": "08h00:08h20" } }, { "rv": null, "creneau": { "id": 2, "mDebut": 20, "mFin": 40, "hFin": 8, "hDebut": 8, "text": "08h20:08h40" } }, { "rv": null, "creneau": { "id": 3, "mDebut": 40, "mFin": 0, "hFin": 9, "hDebut": 8, "text": "08h40:09h00" } }, { "rv": null, "creneau": { "id": 4, "mDebut": 0, "mFin": 20, "hFin": 9, "hDebut": 9, "text": "09h00:09h20" } }, { "rv": null, "creneau": { "id": 5, "mDebut": 20, "mFin": 40, "hFin": 9, "hDebut": 9, "text": "09h20:09h40" } }, { "rv": null, "creneau": { "id": 6, "mDebut": 40, "mFin": 0, "hFin": 10, "hDebut": 9, "text": "09h40:10h00" } }, { "rv": null, "creneau": { "id": 7, "mDebut": 0, "mFin": 20, "hFin": 10, "hDebut": 10, "text": "10h00:10h20" } }, { "rv": null, "creneau": { "id": 8, "mDebut": 20, "mFin": 40, "hFin": 10, "hDebut": 10, "text": "10h20:10h40" } }, { "rv": null, "creneau": { "id": 9, "mDebut": 40, "mFin": 0, "hFin": 11, "hDebut": 10, "text": "10h40:11h00" } }, { "rv": null, "creneau": { "id": 10, "mDebut": 0, "mFin": 20, "hFin": 11, "hDebut": 11, "text": "11h00:11h20" } }, { "rv": null, "creneau": { "id": 11, "mDebut": 20, "mFin": 40, "hFin": 11, "hDebut": 11, "text": "11h20:11h40" } }, { "rv": null, "creneau": { "id": 12, "mDebut": 40, "mFin": 0, "hFin": 12, "hDebut": 11, "text": "11h40:12h00" } }, { "rv": null, "creneau": { "id": 13, "mDebut": 0, "mFin": 20, "hFin": 14, "hDebut": 14, "text": "14h00:14h20" } }, { "rv": null, "creneau": { "id": 14, "mDebut": 20, "mFin": 40, "hFin": 14, "hDebut": 14, "text": "14h20:14h40" } }, { "rv": null, "creneau": { "id": 15, "mDebut": 40, "mFin": 0, "hFin": 15, "hDebut": 14, "text": "14h40:15h00" } }, { "rv": null, "creneau": { "id": 16, "mDebut": 0, "mFin": 20, "hFin": 15, "hDebut": 15, "text": "15h00:15h20" } }, { "rv": null, "creneau": { "id": 17, "mDebut": 20, "mFin": 40, "hFin": 15, "hDebut": 15, "text": "15h20:15h40" } }, { "rv": null, "creneau": { "id": 18, "mDebut": 40, "mFin": 0, "hFin": 16, "hDebut": 15, "text": "15h40:16h00" } }, { "rv": null, "creneau": { "id": 19, "mDebut": 0, "mFin": 20, "hFin": 16, "hDebut": 16, "text": "16h00:16h20" } }, { "rv": null, "creneau": { "id": 20, "mDebut": 20, "mFin": 40, "hFin": 16, "hDebut": 16, "text": "16h20:16h40" } }, { "rv": null, "creneau": { "id": 21, "mDebut": 40, "mFin": 0, "hFin": 17, "hDebut": 16, "text": "16h40:17h00" } }, { "rv": null, "creneau": { "id": 22, "mDebut": 0, "mFin": 20, "hFin": 17, "hDebut": 17, "text": "17h00:17h20" } }, { "rv": null, "creneau": { "id": 23, "mDebut": 20, "mFin": 40, "hFin": 17, "hDebut": 17, "text": "17h20:17h40" } }, { "rv": null, "creneau": { "id": 24, "mDebut": 40, "mFin": 0, "hFin": 18, "hDebut": 17, "text": "17h40:18h00" } }, "jour": "2014-07-06"}, "selectedCreneau": { "rv": null, "creneau": { "id": 1, "mDebut": 0, "mFin": 20, "hFin": 8, "hDebut": 8, "text": "08h00:08h20" } }

```

Cela nécessite une bonne connaissance du code pour être interprété mais lorsque celle-ci est acquise, l'information ci-dessus devient utile pour le débogage. On a surligné ici les éléments du modèle `[$scope.app]` affiché. On rappelle que `[$scope.app]` est la mémoire partagée par les contrôleurs ;

- `[waitingBeforeTask]` : le temps d'attente simulé avant toute requête HTTP ;
- `[debug]` : le mode debug - est forcément *true* si ce bandeau est affiché ;
- `[navbarrun]` : booléen qui contrôle l'affichage de la barre de navigation suivante :



- `[navbarstart]` : booléen qui contrôle l'affichage de la barre de navigation suivante :



- `[errors]` : modèle de la directive `[errors]` ;
- `[view]` : encapsule des informations sur la vue actuellement affichée ;
- `[waiting]` : modèle de la directive `[waiting]` ;
- `[serverUrl, username, password]` : les informations de connexion au service web ;
- `[medecins]` : modèle pour la directive `[[list]]` appliquée aux médecins ;
- `[clients]` : idem pour les clients ;
- `[menu]` : contrôle les options de menu affichées. Celles-ci sont définies dans `[navbar-run.html]` :

```

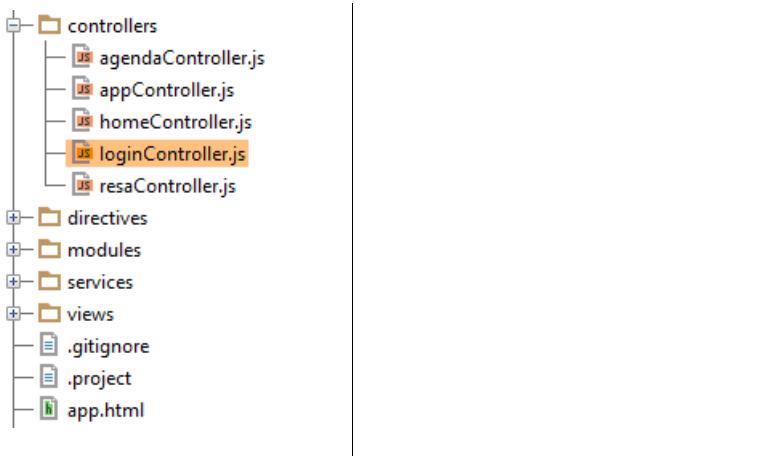
13      <ul class="nav navbar-nav">
14        <li class="active">
15          <a href="">
16            <span translate="options_afficherAgenda" ng-click="home.afficherAgenda()" tooltip="{{'tooltip_afficher_agenda'|translate}}" tooltip-placement="bottom"
17              ng-show="app.menu.afficherAgenda"></span>
18          </a>
19        </li>
20        <li class="active">
21          <a href="">
22            <span translate="options_home" ng-click="agenda.home()" ng-show="app.menu.home"
23                tooltip="{{'tooltip_home'|translate}}" tooltip-placement="bottom"></span>
24          </a>
25        </li>
26        <li class="active">
27          <a href="">
28            <span translate="options_retourAgenda" ng-click="resa.retourAgenda()" ng-show="app.menu.retourAgenda"
29                tooltip="{{'tooltip_retour_agenda'|translate}}" tooltip-placement="bottom"></span></span>
30          </a>
31        </li>
32        <li class="active">
33          <a href="">
34            <span translate="options_validerRv" ng-click="resa.validerRv()" ng-show="app.menu.validatorRv"
35                tooltip="{{'tooltip_valider_rv'|translate}}" tooltip-placement="bottom"></span></span>
36          </a>
37        </li>
38      </ul>
39
40

```

Les options de menu sont aux lignes 16, 23, 29 et 36.

- [formattedJour] : le jour choisi dans le calendrier au format 'aaaa-mm-jj' ;
- [agenda] : l'agenda du médecin. Dans celui-ci, il y a des créneaux libres (rv==null) et réservés. Pour ces derniers, il y alors le nom du client qui a réservé ;
- [selectedCreneau] : le créneau horaire choisi pour faire une réservation ;

3.8.12 Le contrôleur [loginCtrl]



Le contrôleur [loginCtrl] est associé à la vue [views/login.html] qui associée à la page maître produit la page suivante :

RdvMedecins mode debug 3000 http://localhost:8080 admin Connexion Langue Veuillez vous identifier



Cabinet Médical Les Médecins Associés

Authentifiez-vous pour accéder à l'application.

Pour connaître le rôle des différentes saisies, passez le curseur dessus pour avoir une aide.

Le contrôleur [loginCtrl] est le suivant :

```

4   angular.module("rdvmedecins")
5     .controller('loginCtrl', ['$scope', 'config', 'dao', '$location', '$filter', 'utils', '$translate', '$locale',
6       function ($scope, config, dao, $location, $filter, utils, $translate, $locale) {
7
8         // log
9         utils.debug("[loginCtrl] init");
10
11        // -----
12        // on récupère le modèle parent
13        var login = $scope.login;
14        var app = $scope.app;
15        // vue courante
16        app.view = {url: config.urlLogin, model: login, done: false};
17        // on attend la fin du chargement de l'application
18        app.task.action.promise.then(function () {...}, function () {...});
19
20        // -----
21        // méthodes
22
23        // changement de langue
24        login.setLang = function (langKey) {...};
25
26        // authentification
27        login.authenticate = function (serverUrl, username, password) {...};
28      }
29    ]
30  );
31
32;
```

- ligne 13 : [login] sera le modèle de la vue courante ;
- ligne 14 : [app] est la mémoire partagée entre les contrôleurs ;
- ligne 16 : on renseigne [app.view] avec les informations de la vue courante ;

Ce code d'initialisation se retrouvera dans chaque contrôleur. Pour le contrôleur C1 d'une vue V1 ayant le modèle M1 on aura le code d'initialisation suivant :

```

1. var app=$scope.app;
2. var M1=$scope.M1;
3. app.view={url: config.urlV1, model:M1, done:false};
```

- ligne 18 : on se rappelle peut-être que [appCtrl] a lancé une attente simulée référencée par l'objet [app.task.action]. On utilise la [promise] de cette tâche pour attendre sa fin ;
- ligne 39 : la méthode [login.setLang] gère le changement de langues ;
- ligne 47 : la méthode [login.authenticate] gère l'authentification de l'utilisateur ;

Regardons les grands mouvements de la méthode d'authentification :

```

46 // authentification
47 login.authenticate = function (serverUrl, username, password) {
48   utils.debug("[loginCtrl] serverUrl=" + serverUrl + ", username=" + username + ", password=" + password);
49   // on met en route l'attente
50   app.waiting.show = true;
51   app.waiting.title = {text: config.msgWaiting, values: {}};
52   // au départ pas d'erreurs
53   app.errors = {show: false};
54   // attente simulée
55   var task = login.task = {action: utils.waitForSomeTime(app.waitingTimeBeforeTask), isFinished: false};
56   // on charge les médecins
57   var promise = task.action.promise.then(function () {...});
58   // on analyse le résultat et on demande les clients
59   promise = promise.then(function (result) {...});
60   // on analyse le résultat de la demande des clients
61   promise.then(function (result) {...});
62 };
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107

```

- lignes 50-51 : [app.waiting] est le modèle du bandeau d'attente ;
- ligne 53 : [app.errors] est le modèle du bandeau d'erreurs ;
- ligne 55 : une attente simulée est lancée. L'objet [action, isFinished] est référencé par [login.task] et donc puisque [app.view.model=login], par [app.view.model.task]. On rappelle que c'est la condition pour que la tâche puisse être annulée ;
- ligne 57 : après la fin de l'attente simulée, on charge les médecins ;
- ligne 62 : lorsque la demande des médecins a été obtenue, on analyse cette demande. Si les médecins ont été obtenus, on demande alors les clients ;
- ligne 83 : on analyse la réponse obtenue et on affiche la vue finale. Cela se fait avec le code suivant :

```

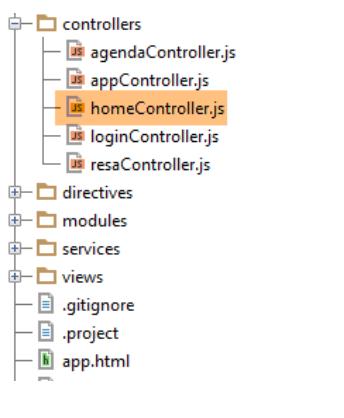
82 // on analyse le résultat de la demande des clients
83 promise.then(function (result) {
84   // fin d'attente
85   app.waiting.show = false;
86   // tâche terminée ?
87   if (task.isFinished) {
88     return;
89   }
90   // on analyse le résultat de la tâche précédente
91   if (result.err == 0) {
92     // on met les clients dans le modèle
93     app.clients = {
94       title: {text: config.listClientsTitle, values: {}},
95       data: result.data, show: true, id: 'clients'};
96     // on passe à la vue suivante
97     app.view.done = true;
98     $location.path(config.urlHome);
99   } else {
100     // il y a eu des erreurs pour obtenir la liste des clients
101     app.errors = {
102       title: {text: config.getClientsErrors, values: {}},
103       messages: utils.getErrors(result, $filter), show: true, model: {}};
104     // tâche terminée
105     task.isFinished = true;
106   }
107 });

```

- ligne 87 : le booléen [task.isFinished] est positionné à *true* dans les cas suivants :
 - l'utilisateur a annulé l'attente ;
 - la demande des médecins s'est terminée avec une erreur ;
- lignes 91-98 : le cas où on a eu les clients ;
- ligne 93 : [app.clients] est le modèle de la directive [list] qui va afficher les clients dans une liste déroulante ;
- lignes 97-98 : on se prépare à changer de vue (ligne 98) mais auparavant on indique que le contrôleur a terminé son travail (ligne 97). On rappelle que [\$scope.app.view.done] est utilisé pour le contrôle de navigation ;

Le point important à noter ici est que les médecins et les clients ont été mis en cache sur le navigateur. Ils ne seront désormais plus demandés au service web.

3.8.13 Le contrôleur [homeCtrl]



Le contrôleur [homeCtrl] est associé à la vue [views/home.html] qui associée à la page maître produit la page suivante :

The screenshot shows the application's main page. At the top, there is a navigation bar with links for "RdvMedecins" and "Agenda". On the right side of the navigation bar are buttons for "mode debug" (with a value of 0), "Déconnexion", and "Langue". The main content area features a logo on the left and the text "Cabinet Médical Les Médecins Associés" in large, bold letters. Below this, a message says "Choisissez un médecin et un jour pour avoir l'agenda". A section titled "Liste des médecins" contains a dropdown menu showing "Mme Marie PELISSIER". At the bottom, there is a date picker labeled "Jour" showing the month of July 2014, with the date "27 08" selected.

La structure du contrôleur [homeCtrl] est la suivante :

```

8 // log
9 utils.debug("[homeCtrl] init");
10
11 // contrôle de navigation
12 var lastView = $scope.app.view;
13 utils.debug("[homeCtrl] lastView", lastView);
14 var navigationAllowed = lastView.url == config.urlHome
15 || lastView.url == config.urlAgenda
16 || (lastView.url == config.urlLogin && lastView.done);
17 if (!navigationAllowed) {
18 // on ne bouge pas
19 $location.path(lastView.url);
20 return;
21 }

```

- lignes 12-20 : c'est le contrôle de navigation. Tous les contrôleurs l'ont sauf [loginCtrl] car la page [/login.html] est accessible sans conditions ;

```

23 // ----- initialisation du modèle de la page
24 // on récupère le modèle parent
25 var home = $scope.home;
26 var app = $scope.app;
27 // vue courante
28 app.view = {url: config.urlHome, model: home, done: false};
29 // préparation du modèle
30 app.navbarstart.show = false;
31 app.navbarrun.show = true;
32 app.errors.show = false;
33 app.titre = {text: config.choixMedecinJourTitle, show: true, model: {}};
34 // le modèle du calendrier
35 var today = new Date();
36 home.datepicker = {jour: today, minDate: today};
37 // le menu
38 app.menu = {afficherAgenda: true};
39 // msg d'attente
40 app.waiting.title = {text: config.msgWaiting, values: {}};
41 app.waiting.show = false;
42 // des infos dans la session ?
43 if (home.jour) {...}

```

- lignes 25-28 : on retrouve là des lignes analogues à celles rencontrées dans le contrôleur [loginCtrl]. [home] est ainsi le modèle de la vue associée au contrôleur ;
- ligne 33 : un attribut que nous n'avions pas encore rencontré. C'est le modèle du bandeau de titre de la vue :

Choisissez un médecin et un jour pour avoir l'agenda

- ligne 36 : [home.datepicker] est le modèle du calendrier ;
- ligne 38 : [app.menu] est le modèle du menu de la barre de navigation. Ici l'option [Agenda] sera présente. C'est elle qui permet de demander l'agenda d'un médecin ;

Enfin, le contrôleur a deux méthodes :

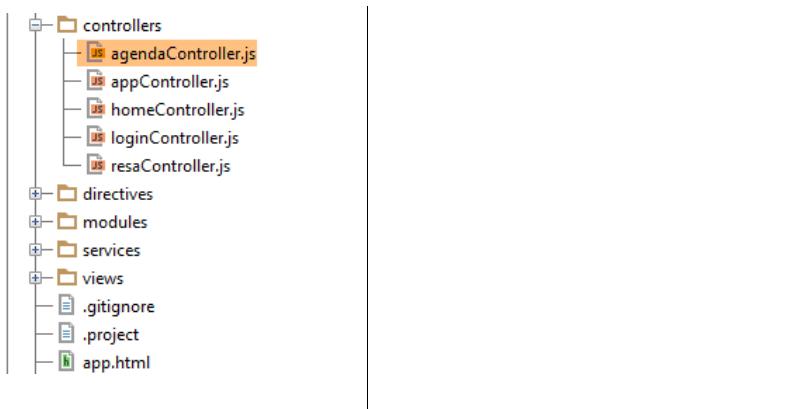
```

48 // ----- méthodes du modèle
49
50 // changement de langue
51 home.setLang = function (langKey) {...};
52
53 // affichage agenda
54 home.afficherAgenda = function afficherAgenda() {...};
55
56
57
58
59
60
61

```

L'affichage de l'agenda (ligne 51) a été traité au paragraphe 3.7.8, page 216.

3.8.14 Le contrôleur [agendaCtrl]



Le contrôleur [agendaCtrl] est associé à la vue [views/agenda.html] qui associe à la page maître produit la page suivante :

Rendez - vous de Mme Marie PELISSIER le 2014-07-06

Créneau horaire	Client	Action
08h00:08h20		<button>Réserver</button> Faire une réservation
08h20:08h40		<button>Réserver</button>

La structure du contrôleur [agendaCtrl] est la suivante :

```
8 // log
9 utils.debug("[agendaCtrl] init");
10 // contrôle de navigation
11 var lastView = $scope.app.view;
12 utils.debug("[agendaCtrl] lastView", lastView);
13 var navigationAllowed = lastView.url == config.urlAgenda
14 || lastView.url == config.urlResa
15 || (lastView.url == config.urlHome && lastView.done);
16 if (!navigationAllowed) {
17 // on ne bouge pas
18 $location.path(lastView.url);
19 return;
20 }
```

- les lignes 10-20 assurent le contrôle de navigation ;

```

21 // ----- modèle de la page
22 // on récupère le modèle parent
23 var agenda = $scope.agenda;
24 var app = $scope.app;
25 // vue courante
26 app.view = {url: config.urlAgenda, model: agenda, done: false};
27 // les barres de navigation
28 app.navbarstart.show = false;
29 app.navbarrun.show = true;
30 // le message d'attente
31 app.waiting.show = false;
32 app.waiting.title = {text: config.msgWaiting, values: {}};
33 // le message d'erreurs
34 app.errors.show = false;
35 // le titre de la vue
36 app.titre = {
37   text: config.agendaTitre,
38   model: {
39     titre: app.agenda.medecin.titre,
40     prenom: app.agenda.medecin.prenom,
41     nom: app.agenda.medecin.nom,
42     jour: app.agenda.jour},
43   show: true
44 };
45 // le menu de la vue
46 app.menu = {home: true};

```

- lignes 23-26 : [agenda] sera le modèle de la vue associée au contrôleur [agendaCtrl] ;
- lignes 36-44 : [app.titre] est le modèle du bandeau de titre suivant :

Rendez - vous de Mme Marie PELISSIER le 2014-07-06

- ligne 46 : le menu aura l'option [Home / Accueil] :



Les méthodes du contrôleur sont les suivantes :

```

56 // ----- méthodes de la page
57
58 // changement de langue
59 agenda.setLang = function (langKey) {...};
60
61 // retour à la vue [home]
62 agenda.home = function () {...};
63
64 // réservation d'un créneau
65 agenda.reserver = function (creneauId) {...};
66
67 // suppression d'un rendez-vous
68 agenda.supprimer = function (idRv) {...}
69
70
71
72
73
74
75

```

- ligne 95 : la méthode [agenda.supprimer] a été traitée au paragraphe 3.7.9, page 225 ;

La méthode [agenda.home] est une méthode de pure navigation :

```

66 // retour à la vue [home]
67 agenda.home = function () {
68   app.view.done = true;
69   $location.path(config.urlHome)
70 };

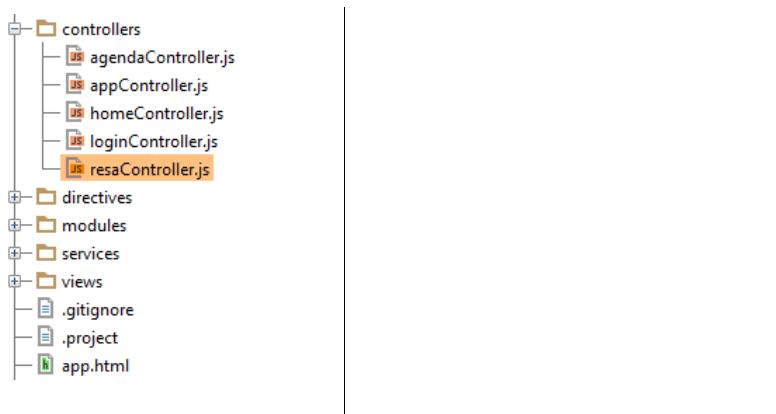
```

La méthode [agenda.reserver] est la suivante :

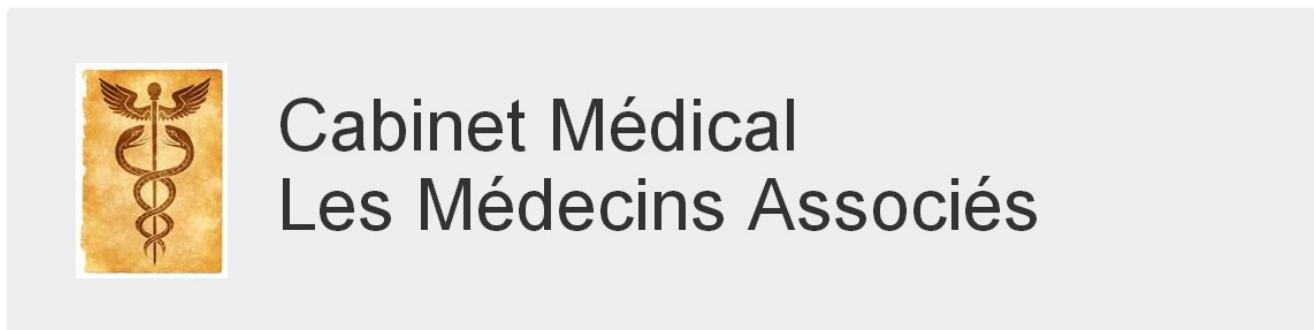
```
47      // mise en forme des créneaux
48      var creneauxMedecin = app.agenda.creneauxMedecin;
49      angular.forEach(creneauxMedecin, function (creneauMedecin) {...});
50
51      // ----- méthodes de la page
52
53      // changement de langue
54      agenda.setLang = function (langKey) {...};
55
56      // retour à la vue [home]
57      agenda.home = function () {...};
58
59      // réservation d'un créneau
60      agenda.reserver = function (creneauId) {
61          // réservation d'un nouveau créneau
62          utils.debug("[agendaCtrl] réserver crenauId", crenauId);
63          // on recherche dans le modèle le créneau concerné
64          var done = false;
65          for (var i = 0; i < creneauxMedecin.length && !done; i++) {
66              var creneauMedecin = creneauxMedecin[i];
67              if (creneauMedecin.id == crenauId) {
68                  // on met le créneau dans le modèle
69                  app.selectedCreneau = creneauMedecin;
70                  // trouvé
71                  done = true;
72              }
73          }
74          utils.debug("[agendaCtrl] selectedCreneau", app.selectedCreneau);
75          // on passe à la page de réservation
76          app.view.done = true;
77          $location.path(config.urlResa);
78      };
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
```

- ligne 78 : le paramètre de la fonction [reserver] est le n° du créneau (id) ;
- lignes 83-90 : visent à retrouver le créneau horaire ayant cet identifiant ;
- ligne 87 : le créneau trouvé est mis dans la mémoire partagée [app]. Le contrôleur [resaCtrl] qui va prendre la main (ligne 95) va exploiter cette information pour afficher son bandeau de titre ;
- lignes 94-95 : navigation vers [/resa.html] ;

3.8.15 Le contrôleur [resaCtrl]



Le contrôleur [resaCtrl] est associé à la vue [views/reso.html] qui associée à la page maître produit la page suivante :



Cabinet Médical Les Médecins Associés

Rendez - vous de Mme Marie PELISSIER le 2014-07-06 à 08h00:08h20

Liste des clients

Mr Jules MARTIN ▾

La structure du contrôleur [resaCtrl] est la suivante :

```

8   // log
9     utils.debug("[resaCtrl] init");
10
11   // contrôle de navigation
12   var lastView = $scope.app.view;
13   utils.debug("[resaCtrl] lastView", lastView);
14   var navigationAllowed = lastView.url == config.urlResa
15   || (lastView.url == config.urlAgenda && lastView.done);
16   if (!navigationAllowed) {
17     // on ne bouge pas
18     $location.path(lastView.url);
19     return;
20   }

```

- lignes 12-20 : le contrôle de navigation ;

```

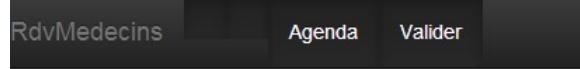
22   // ----- initialisation vue
23   // on récupère le modèle parent
24   var resa = $scope.resa;
25   var app = $scope.app;
26   // vue courante
27   app.view = {url: config.urlResa, model: resa, done: false};
28   // ----- modèle de la page
29   // les barres de navigation
30   app.navbarstart.show = false;
31   app.navbarrun.show = true;
32   // le message d'attente
33   app.waiting.show = false;
34   app.waiting.title = {text: config.msgWaiting, values: {}};
35   // le message d'erreurs
36   app.errors.show = false;
37   // le titre de la page
38   app.titre = {
39     text: config.resaTitre, model: {titre: app.agenda.medecin.titre,
40     prenom: app.agenda.medecin.prenom,
41     nom: app.agenda.medecin.nom,
42     jour: app.agenda.jour,
43     creneau: app.selectedCreneau.creneau.text},
44     show: true
45   };
46   // le menu de la page
47   app.menu = {retourAgenda: true, validerRv: true};

```

- lignes 24-27 : [resa] sera le modèle de la vue courante ;
- lignes 38-45 : [app.titre] est le modèle du bandeau de titre suivant :

Rendez - vous de Mme Marie PELISSIER le 2014-07-06 à 08h00:08h20

- ligne 47 : deux options de menu sont affichées :



Les méthodes du contrôleur sont les suivantes :

```

50 // -----
51 // changement de langue
52 resa.setLang = function (langKey) {...};

53 resa.retourAgenda = function () {
54   app.view.done = true;
55   $location.path(config.urlAgenda);
56 };

57 // validation d'un rendez-vous
58 resa.validerRv = function () {...}
59
60
61
62
63
64
65

```

La méthode [resa.valider] a été étudiée au paragraphe 3.7.9, page 225.

3.8.16 La gestion des langues

Tous les contrôleurs offrent la méthode [setLang] suivante :

```

51 // -----
52 resa.setLang = function (langKey) {
53   // on change la langue courante
54   $translate.use(langKey);
55   // changement de locale pour le calendrier
56   angular.copy(config.locales[langKey], $locale);
57 };

```

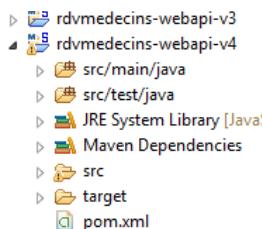
Elle aurait pu être factorisée dans le contrôleur [appCtrl].

4 Exploitation de l'application

Nous souhaitons maintenant exploiter l'application en-dehors des IDE STS (pour le serveur) et Webstorm (pour le client).

4.1 Déploiement du service web sur un serveur Tomcat

Nous avons vu au paragraphe 2.11.9, page 68, comment créer une archive war pour Tomcat. Nous répétons l'opération ici. Tout d'abord, pour préserver l'existant, nous dupliquons le projet Eclipse [rdvmedecins-webapi-v3] dans [rdvmedecins-webapi-v4].



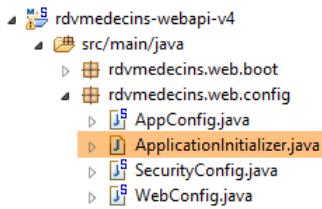
Le fichier [pom.xml] est modifié de la façon suivante :

```
1. <modelVersion>4.0.0</modelVersion>
2. <groupId>istia.st.spring4.mvc</groupId>
3. <artifactId>rdvmedecins-webapi-v4</artifactId>
4. <version>0.0.1-SNAPSHOT</version>
5. <packaging>war</packaging>
6.
7. <name>rdvmedecins-webapi-v3</name>
8. <description>Gestion de RV Médecins</description>
9. <parent>
10.   <groupId>org.springframework.boot</groupId>
11.   <artifactId>spring-boot-starter-parent</artifactId>
12.   <version>1.0.0.RELEASE</version>
13. </parent>
14. <dependencies>
15.   <dependency>
16.     <groupId>org.springframework.boot</groupId>
17.     <artifactId>spring-boot-starter-web</artifactId>
18.   </dependency>
19.   <dependency>
20.     <groupId>org.springframework.boot</groupId>
21.     <artifactId>spring-boot-starter-security</artifactId>
22.   </dependency>
23.   <dependency>
24.     <groupId>org.springframework.boot</groupId>
25.     <artifactId>spring-boot-starter-tomcat</artifactId>
26.     <scope>provided</scope>
27.   </dependency>
28.   <dependency>
29.     <groupId>istia.st.spring4.rdvmedecins</groupId>
30.     <artifactId>rdvmedecins-metier-dao-v2</artifactId>
31.     <version>0.0.1-SNAPSHOT</version>
32.   </dependency>
33. </dependencies>
```

Les modifications sont à faire à deux endroits :

- ligne 5 : il faut indiquer qu'on va générer une archive war (Web ARchive) ;
- lignes 23-27 : il faut ajouter une dépendance sur l'artifact [spring-boot-starter-tomcat]. Cet artifact amène toutes les classes de Tomcat dans les dépendances du projet ;
- ligne 26 : cet artifact est [provided], c-à-d que les archives correspondantes ne seront pas placées dans le war généré. En effet, ces archives seront trouvées sur le serveur Tomcat sur lequel s'exécutera l'application ;

Il faut par ailleurs configurer l'application web. En l'absence de fichier [web.xml], cela se fait avec une classe héritant de [SpringBootServletInitializer] :



La classe [ApplicationInitializer] est la suivante :

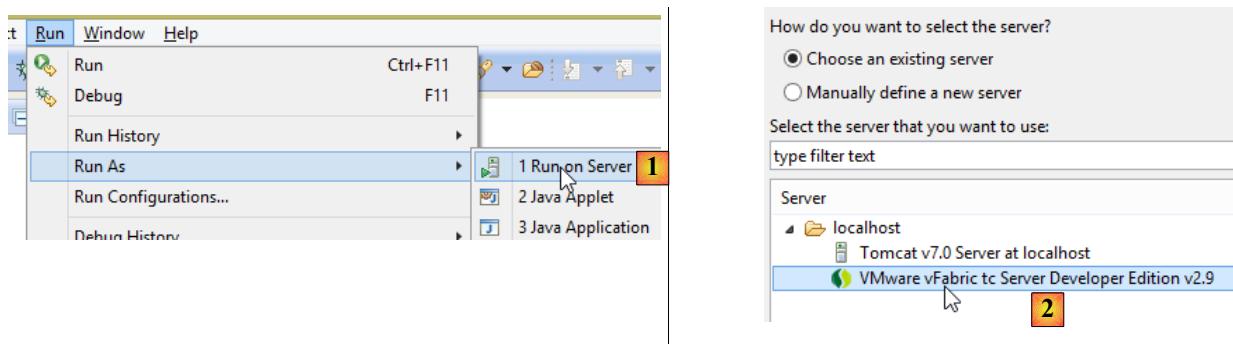
```

1. package rdvmedecins.web.config;
2.
3. import org.springframework.boot.builder.SpringApplicationBuilder;
4. import org.springframework.boot.context.web.SpringBootServletInitializer;
5.
6. public class ApplicationInitializer extends SpringBootServletInitializer {
7.     @Override
8.     protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
9.         return application.sources(AppConfig.class);
10.    }
11.
12. }
```

- ligne 6 : la classe [ApplicationInitializer] étend la classe [SpringBootServletInitializer] ;
- ligne 8 : la méthode [configure] est redéfinie (ligne 7) ;
- ligne 9 : on fournit la classe [AppConfig] qui configure le projet ;

Ceci fait, il peut être nécessaire de mettre à jour le projet Maven (j'ai du le faire) : [clic droit sur projet / Maven / Update project] ou [Alt-F5].

Pour exécuter le projet, on peut procéder ainsi :



- en [1], on exécute le projet sur l'un des serveurs enregistrés dans l'IDE Eclipse ;
- en [2], on choisit [tc Server Developer] qui est présent par défaut. C'est une variante de Tomcat ;

On obtient le résultat suivant :

VMware vFabric tc Runtime 2.9.4.RELEASE/7.0.47.ARELEASE - Rapport d'erreur

http://localhost:8080/rdvmedecins-webapi-v4/

Etat HTTP 404 -

type Rapport d'état

message

description La ressource demandée n'est pas disponible.

VMware vFabric tc Runtime 2.9.4.RELEASE/7.0.47.ARELEASE

C'est normal. Rappelons que le service web n'a pas l'URL [/] dans ses méthodes. Lorsqu'on essaie l'URL [/getAllMedecins], on a la réponse suivante :

VMware vFabric tc Runtime

localhost:8080/rdvmedecins-webapi-v4/getAllMedecins

Etat HTTP 404 - /ge

type Rapport d'état

message /getAllMedecins

description La ressource demandée n'

VMware vFabric tc Runtime 2.9

Authentification requise

Le serveur http://localhost:8080 requiert un nom d'utilisateur et un mot de passe. Message du serveur : Realm.

Nom d'utilisateur :

Mot de passe :

Se connecter Annuler

C'est normal. Le service web est protégé.

Maintenant lançons le client [rdvmedecins-angular-v2] dans Webstorm :

mode debug 3000 180/rdvmedecins-webapi-v4 admin ---- Connexion Langue Veuillez vous identifier

En [1], on met l'URL du nouveau service web [http://localhost:8080/rdvmedecins-webapi-v4]. On obtient le résultat suivant :



Cabinet Médical Les Médecins Associés

Choisissez un médecin et un jour pour avoir l'agenda

Liste des médecins

Mme Marie PELISSIER ▾

Jour



Pour exécuter l'application en-dehors de l'IDE STS, il existe diverses solutions. En voici une.

Téléchargez une version de Tomcat [<http://tomcat.apache.org/download-80.cgi>] (juillet 2014) :

8.0.9

Please see the [README](#) file for packaging information. It explains what

Binary Distributions

- Core:
 - [zip \(pgp, md5\)](#)
 - [tar.gz \(pgp, md5\)](#)
 - [32-bit Windows zip \(pgp, md5\)](#)
 - [64-bit Windows zip \(pgp, md5\)](#)
 - [64-bit Itanium Windows zip \(pgp, md5\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, md5\)](#)
- Full documentation:
 - [tar.gz \(pgp, md5\)](#)

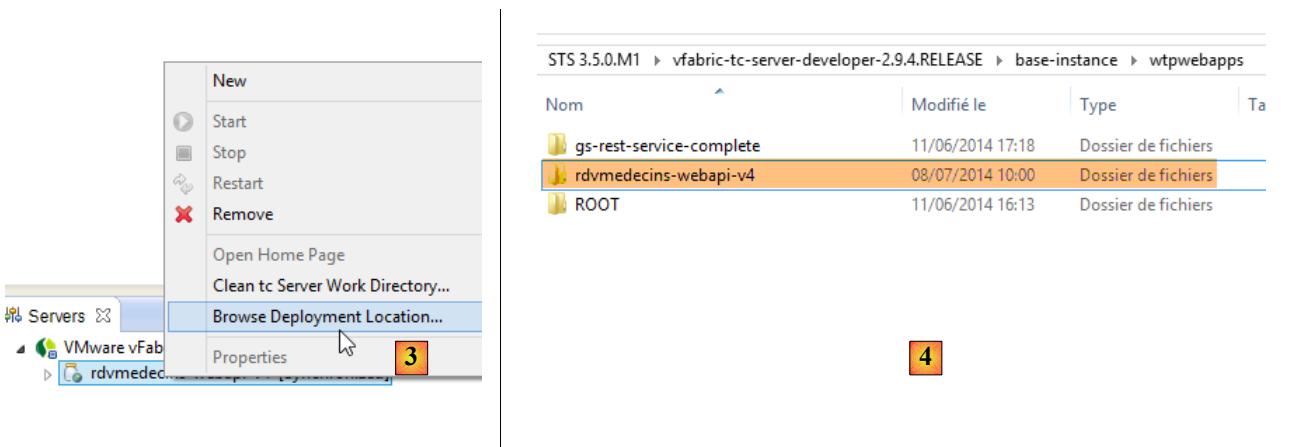
[1]

2

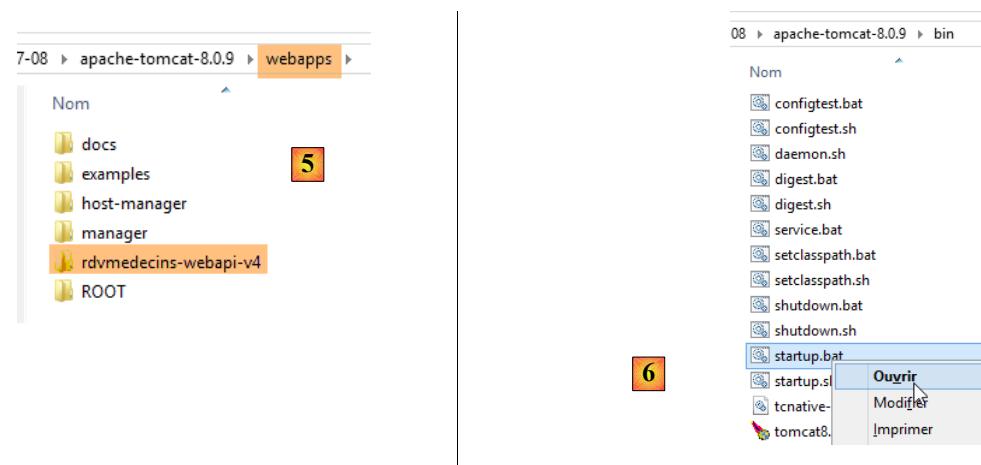
apache-tomcat-8.0.9

Nom
bin
conf
lib
logs
temp
webapps
work
LICENSE
NOTICE
RELEASE-NOTES
RUNNING.txt

On choisit en [1] une version zippée qu'on dézippe en [2]. On revient dans STS :

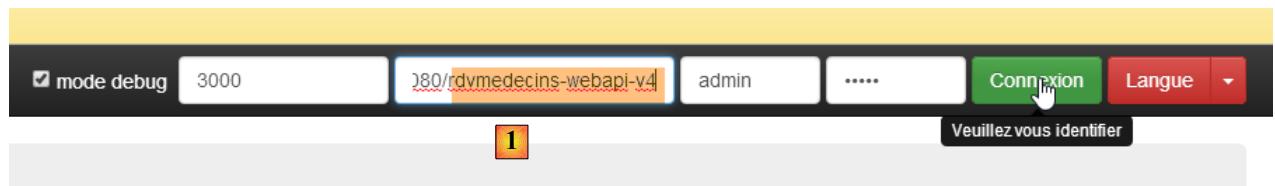


- dans l'onglet [Servers], on clique droit sur l'application [rdvmedecins-webapi-v4] et on sélectionne l'option [Browse Deployment Location] ;
- en [4] : on copie le dossier [rdvmedecins-webapi-v4] ;



- en [5], on colle le dossier [rdvmedecins-webapi-v4] dans le dossier [webapps] de Tomcat ;
- en [6], on exécute le fichier de commande [startup.bat] (le serveur Tomcat intégré à STS doit lui être arrêté). Une fenêtre DOS s'ouvre pour afficher les logs de Tomcat. Ils doivent montrer que l'application [rdvmedecins-webapi-v4] a été lancée.

Pour le vérifier, on exécute de nouveau le client Angular [rdvmedecins-angular-v2] dans Webstorm :



En [1], on met l'URL du nouveau service web [<http://localhost:8080/rdvmedecins-webapi-v4>]. On obtient le résultat suivant :



Cabinet Médical Les Médecins Associés

Choisissez un médecin et un jour pour avoir l'agenda

Liste des médecins

Mme Marie PELISSIER ▾

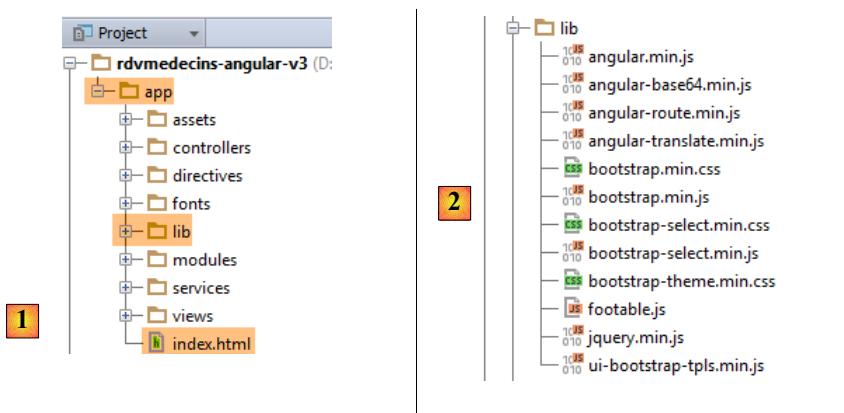
Jour



4.2 Déploiement du client Angular sur le serveur Tomcat

Maintenant que le service web a été déployé sur Tomcat, nous allons maintenant déployer le client Angular sur un serveur lui aussi. Ce peut très bien être le serveur qui héberge déjà le service web. Nous prenons cette voie.

Tout d'abord nous dupliquons le client [rdvmedecins-angular-v2] dans [rdvmedecins-angular-v3] et nous faisons les modifications suivantes :



- en [1], tout a été déplacé dans un dossier [app] ;
- en [1], on a supprimé le dossier [bower-components] qui contenait les diverses bibliothèques CSS et JS nécessaires au projet. Tous ces éléments ont été recopier dans le dossier [lib] [2] ;
- en [1], le fichier [app.html] a été renommé [index.html] ;

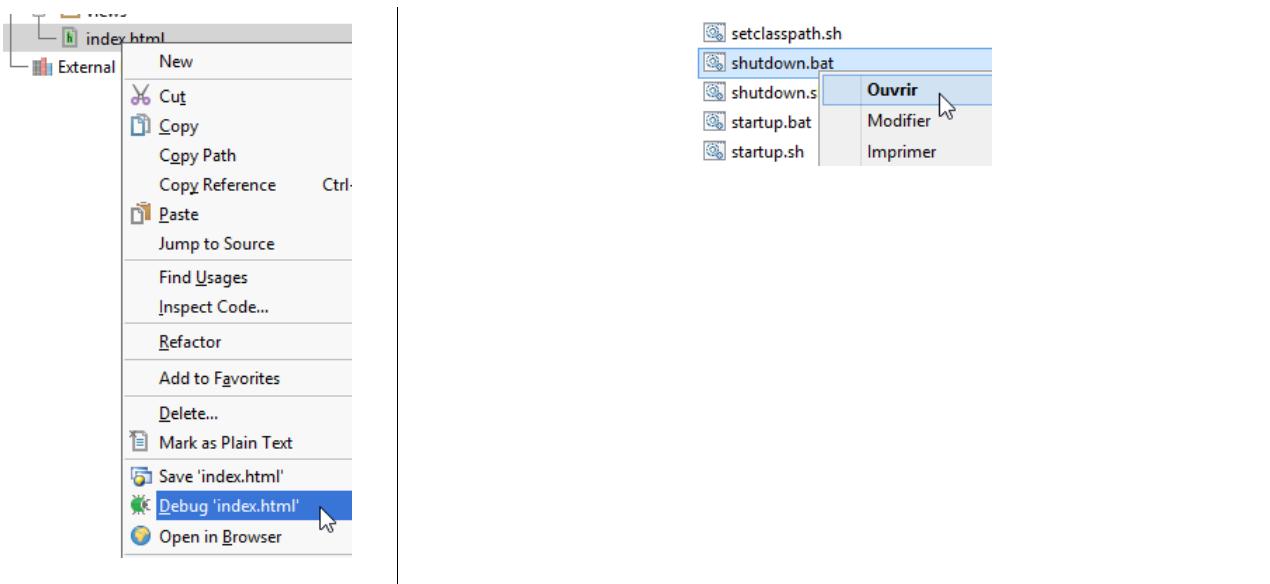
Le fichier [index.html] a été modifié pour prendre en compte les changements de chemin des ressources utilisées :

```
1. <!DOCTYPE html>
2. <html ng-app="rdvmedecins">
3. <head>
4.   <title>RdvMedecins</title>
5. ...
6.   <!-- le CSS -->
7. ...
8.   <link href="Lib/bootstrap-theme.min.css" rel="stylesheet"/>
9.   <link href="Lib/bootstrap-select.min.css" rel="stylesheet"/>
10. </head>
11. <!-- contrôleur [appCtrl], modèle [app] -->
12. <body ng-controller="appCtrl">
13. <div class="container">
14. ...
15. </div>
16. <!-- Bootstrap core JavaScript ===== -->
17. <script type="text/javascript" src="Lib/jquery.min.js"></script>
18. <script type="text/javascript" src="Lib/bootstrap.min.js"></script>
19. <script type="text/javascript" src="Lib/bootstrap-select.min.js"></script>
20. <script type="text/javascript" src="Lib/footable.js"></script>
21. <!-- angular js -->
22. <script type="text/javascript" src="Lib/angular.min.js"></script>
23. <script type="text/javascript" src="Lib/ui-bootstrap-tpls.min.js"></script>
24. <script type="text/javascript" src="Lib/angular-route.min.js"></script>
25. <script type="text/javascript" src="Lib/angular-translate.min.js"></script>
26. <script type="text/javascript" src="Lib/angular-base64.min.js"></script>
27. <!-- modules -->
28. ...
29. <!-- services -->
30. ...
31. <!-- directives -->
32. ...
33. <!-- controllers -->
34. ....
35. </body>
36. </html>
```

Par ailleurs, le contrôleur [loginCtrl] a été modifié pour pointer sur le bon serveur afin d'éviter à l'utilisateur de taper son URL :

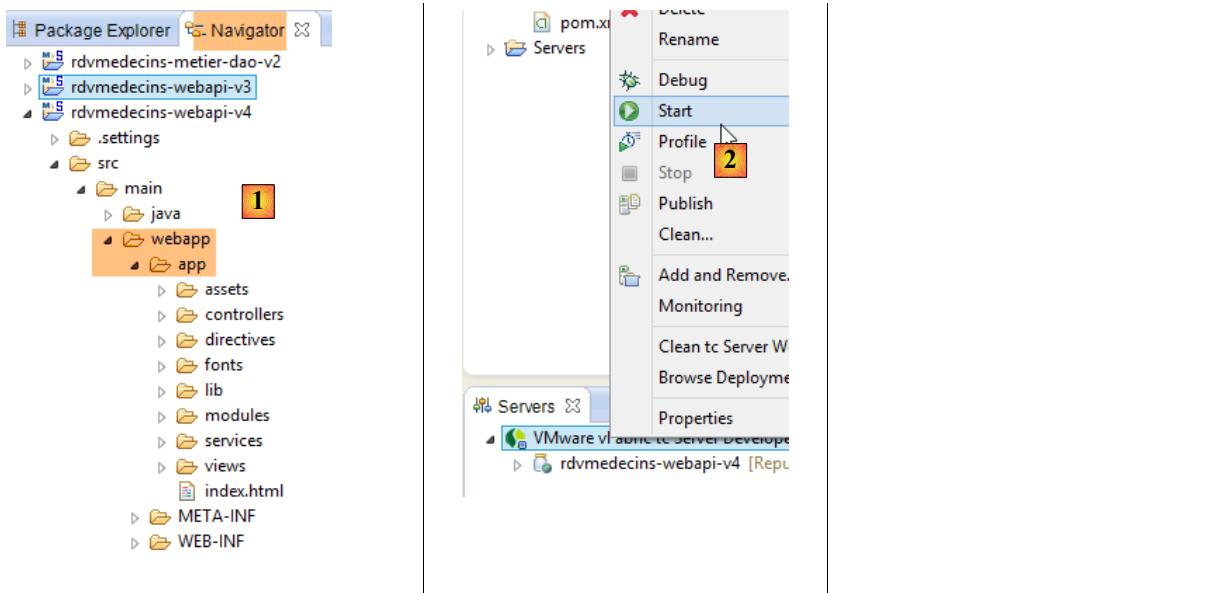
```
1. // credentials
2. app.serverUrl = "http://localhost:8080/rdvmedecins-webapi-v4";
3. app.username = "admin";
4. app.password = "admin";
```

Ceci fait, exécutons le fichier [index.html] :

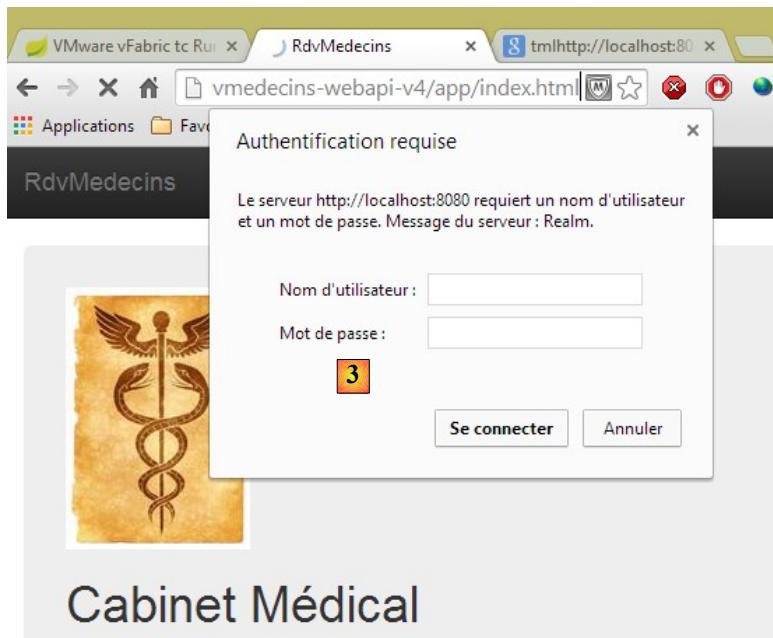


Puis connectons-nous au service web. Ca doit marcher. Ceci vérifié, arrêtons le serveur Tomcat. Nous allons réutiliser le serveur intégré de STS.

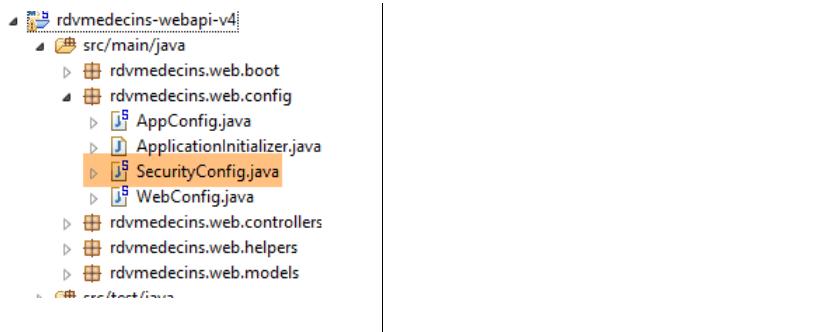
Dans STS, copions tout le contenu du dossier [rdvmedecins-angular-v3/app] dans le dossier [webapp] du projet [rdvmedecins-webapi-v4] (onglet *Navigator*) [1] :



Ceci fait, lançons [2], le serveur VMware de STS, puis demandons l'URL [<http://localhost:8080/rdvmedecins-webapi-v4/app/index.html>] :



On a un problème de droits en [3]. Ce n'est pas étonnant car on a protégé le service web. Il faut qu'on déclare que l'accès au fichier [/app/index.html] est libre. Revenons dans Eclipse :



On se rappelle que les droits d'accès ont été déclarés dans la classe [SecurityConfig]. Modifions celle-ci de la façon suivante :

```

1.   @Override
2.   protected void configure(HttpSecurity http) throws Exception {
3.       // CSRF
4.       http.csrf().disable();
5.       // le mot de passe est transmis par le header Authorization: Basic xxxx
6.       http.httpBasic();
7.       // la méthode HTTP OPTIONS doit être autorisée pour tous
8.       http.authorizeRequests() //
9.           .antMatchers(HttpMethod.OPTIONS, "/", "/**").permitAll();
10.      // le dossier [app] est accessible à tous
11.      http.authorizeRequests() //
12.          .antMatchers(HttpMethod.GET, "/app", "/app/**").permitAll();
13.      // seul le rôle ADMIN peut utiliser l'application
14.      http.authorizeRequests() //
15.          .antMatchers("/", "/**") // toutes les URL
16.          .hasRole("ADMIN");
17. }

```

- lignes 11-12 : on autorise tout le monde à lire le dossier [app] et son contenu. On s'inspire, pour le faire, des lignes précédentes.

Maintenant, relançons le serveur Tomcat de STS puis demandons de nouveau l'URL [<http://localhost:8080/rdvmedecins-webapi-v4/app/index.html>] :



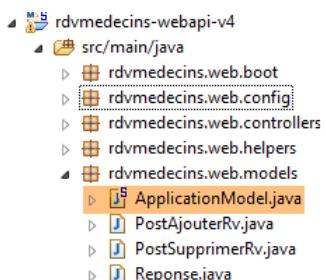
Cette fois-ci, c'est bon.

4.3 Les entêtes CORS

On se souvient peut-être que nous avons bataillé dur pour gérer les entêtes CORS. Dans l'exemple précédent :

- le service web est à l'URL [<http://localhost:8080/rdvmedecins-webapi-v4>];
- le client HTML est à l'URL [<http://localhost:8080/rdvmedecins-webapi-v4/app>];

Le client HTML et le service web sont donc sur le même serveur [<http://localhost:8080>]. Il n'y a alors pas de conflits CORS car ceux-ci n'interviennent que lorsque le client et le serveur ne sont pas dans le même domaine. On devrait pouvoir le vérifier. Nous revenons dans STS :



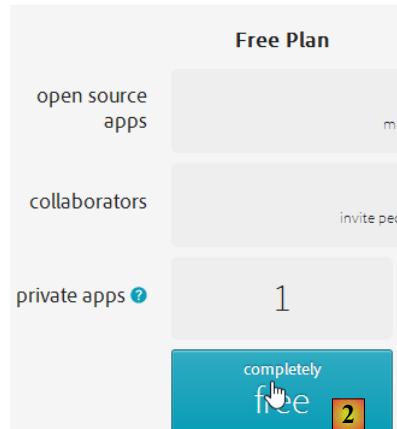
La génération ou non des entêtes CORS est contrôlée par un booléen défini dans la classe [ApplicationModel] :

```
// données de configuration
private boolean CORSneeded = true;
```

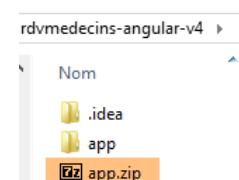
Nous passons le booléen ci-dessus à **false**, nous relançons le service web et nous redemandons l'URL [<http://localhost:8080/rdvmedecins-webapi-v4/app/index.html>]. On constate que l'application fonctionne.

4.4 Déploiement du client Angular sur une tablette Android

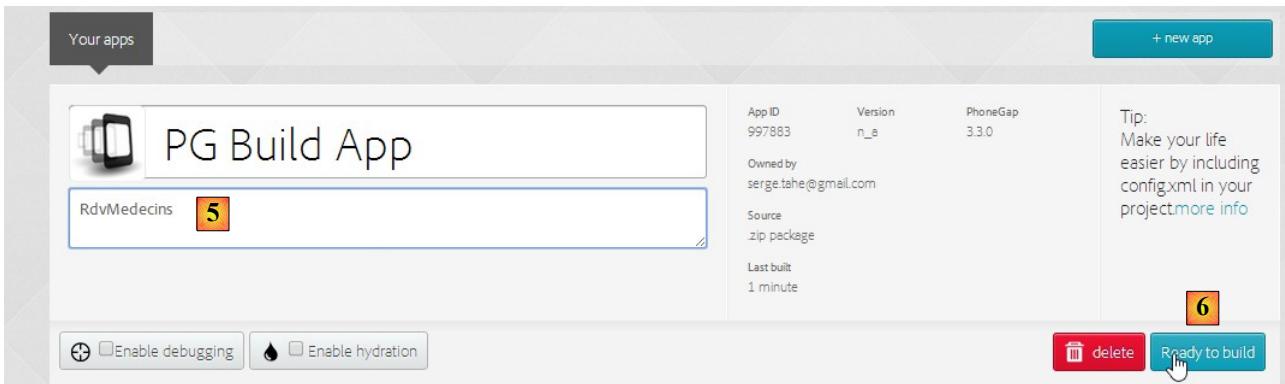
L'outil [Phonegap] [<http://phonegap.com/>] permet de produire un exécutable pour mobile (Android, IoS, Windows 8, ...) à partir d'une application HTML / JS / CSS. Il y a différentes façons d'arriver à ce but. Nous utilisons le plus simple : un outil présent en ligne sur le site de Phonegap [<http://build.phonegap.com/apps>].



- avant [1], vous aurez peut-être à créer un compte ;
- en [1], on démarre ;
- en [2], on choisit un plan gratuit n'autorisant qu'une application Phonegap ;



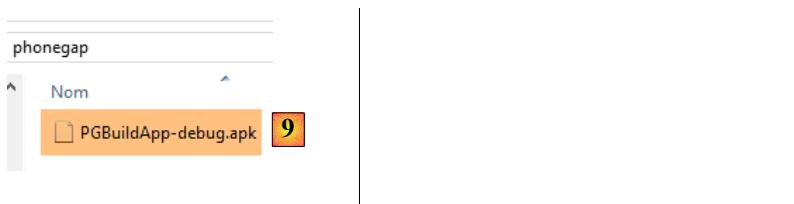
- en [3], on télécharge l'application zippée [4] (le dossier [app] créé page 296 est zippé) ;



- en [5], on donne un nom à l'application ;
- en [6], on la construit. Cette opération peut prendre 1 minute. Patientez jusqu'à ce que les icônes des différentes plateformes mobiles indiquent que la construction est terminée ;



- seuls les binaires Android [7] et Windows [8] ont été générés ;
- on clique sur [7] pour télécharger le binaire d'Android ;



- en [9] le binaire [apk] téléchargé ;

Lancez un émulateur [GenyMotion] pour une tablette Android (voir paragraphe 6.4, page 313) :



Ci-dessus, on lance un émulateur de tablette avec l'API 16 d'Android. Une fois l'émulateur lancé,

- déverrouillez-le en tirant le verrou (s'il est présent) sur le côté puis en le lâchant ;
- avec la souris, tirez le fichier [PGBuildApp-debug.apk] que vous avez téléchargé et déposez-le sur l'émulateur. Il va être alors installé et exécuté ;



Il faut changer l'URL en [1]. Pour cela, dans une fenêtre de commandes, tapez la commande [ipconfig] (ligne 1 ci-dessous) qui va afficher les différentes adresses IP de votre machine :

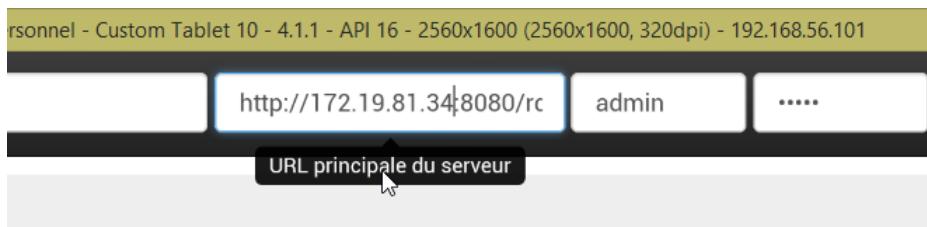
```

1. C:\Users\Serge Tahé>ipconfig
2.
3. Configuration IP de Windows
4.
5.
6. Carte réseau sans fil Connexion au réseau local* 15 :
7.
8.     Statut du média. . . . . : Média déconnecté
9.     Suffrage DNS propre à la connexion. . . :
10.
11. Carte Ethernet Connexion au réseau local :
12.
13.     Suffrage DNS propre à la connexion. . . : ad.univ-angers.fr
14.     Adresse IPv6 de liaison locale. . . . . : fe80::698b:455a:925:6b13%4
15.     Adresse IPv4. . . . . : 172.19.81.34
16.     Masque de sous-réseau. . . . . : 255.255.0.0
17.     Passerelle par défaut. . . . . : 172.19.0.254
18.
19. Carte réseau sans fil Wi-Fi :
20.

```

21. Statut du média. : Média déconnecté
22. Suffixe DNS propre à la connexion. . . :
- 23.
24. ...

Notez soit l'adresse IP Wifi (lignes 6-9), soit l'adresse IP sur le réseau local (lignes 11-17). Puis utilisez cette adresse IP dans l'URL du serveur web :



Ceci fait, connectez-vous au service web :

Liste des médecins

Mme Marie PELISSIER ▾

Jour



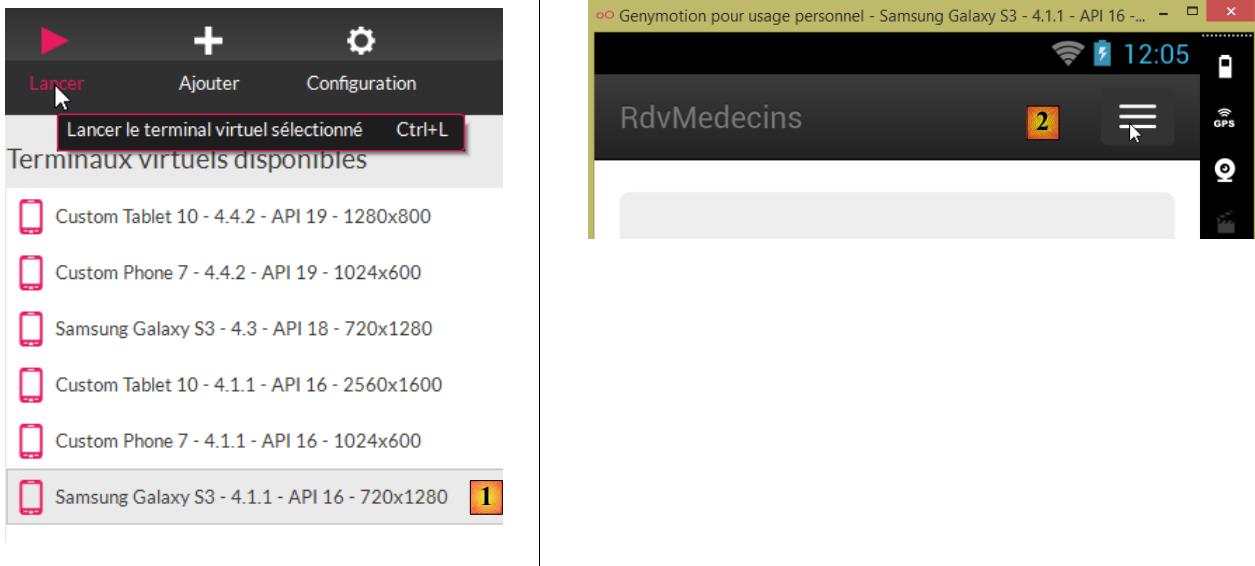
Testez l'application sur l'émulateur. Elle doit fonctionner. Côté serveur, on peut ou non autoriser les entêtes CORS dans la classe [ApplicationModel] :

```
// données de configuration
private boolean CORSneeded = false;
```

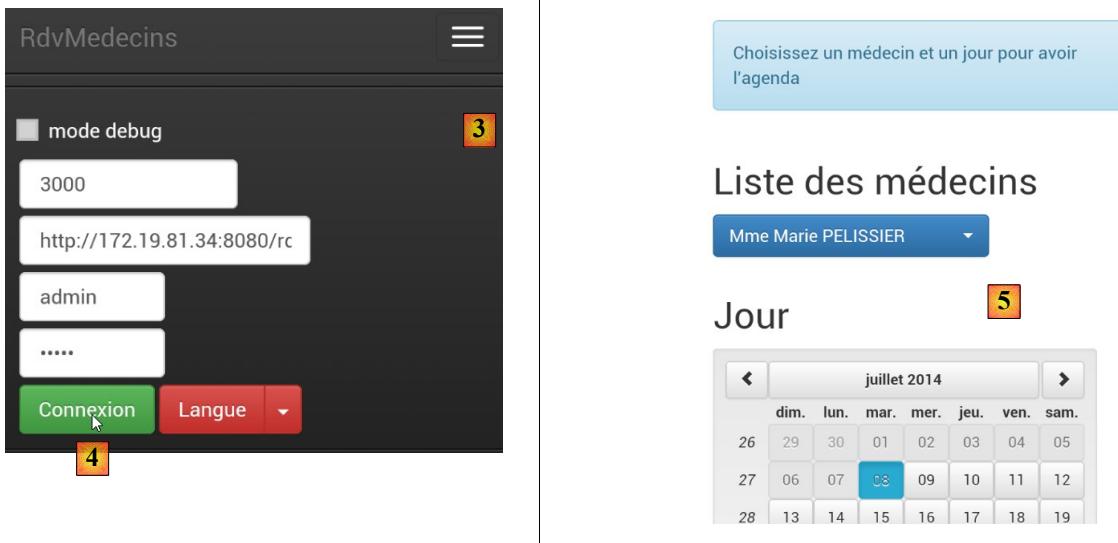
Cela n'a pas d'importance pour l'application Android. Celle-ci ne s'exécute pas dans un navigateur. Or l'exigence des entêtes CORS vient du navigateur et non pas du serveur.

4.5 Déploiement du client Angular sur l'émulateur d'un smartphone Android

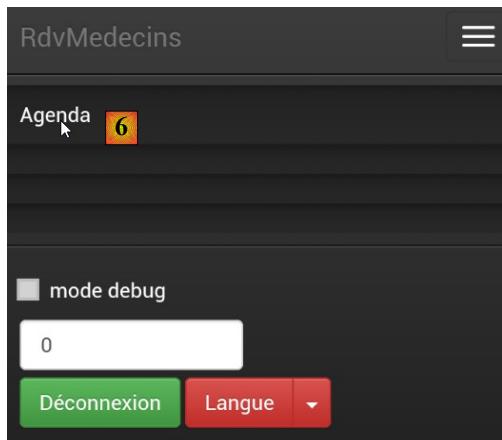
On répète l'opération précédente avec un émulateur pour smartphone. On veut vérifier comment se comporte notre client sur des petits écrans :



- en [1], on lance un émulateur de smartphone ;
- en [2] et [3], la barre de navigation a été repliée dans un menu ;



- en [4], on se connecte ;
- en [5], la liste et le calendrier sont l'un sous l'autre au lieu d'être l'un à côté de l'autre ;



Rendez - vous de Mme Marie PELISSIER le
2014-07-08

Créneau horaire	Action
+ 08h00:08h20	Réserver
7 + 08h20:08h40	Réserver

- en [6], on demande l'agenda ;
- en [7], l'écran étant trop petit, les créneaux ont une partie cachée. C'est la bibliothèque [footable] qui a fait ce travail ;

Rendez - vous de Mme Marie PELISSIER le
2014-07-08

Créneau horaire	Action
- 08h00:08h20	Supprimer
Client: Mr Jules MARTIN	
+ 08h20:08h40	Réserver

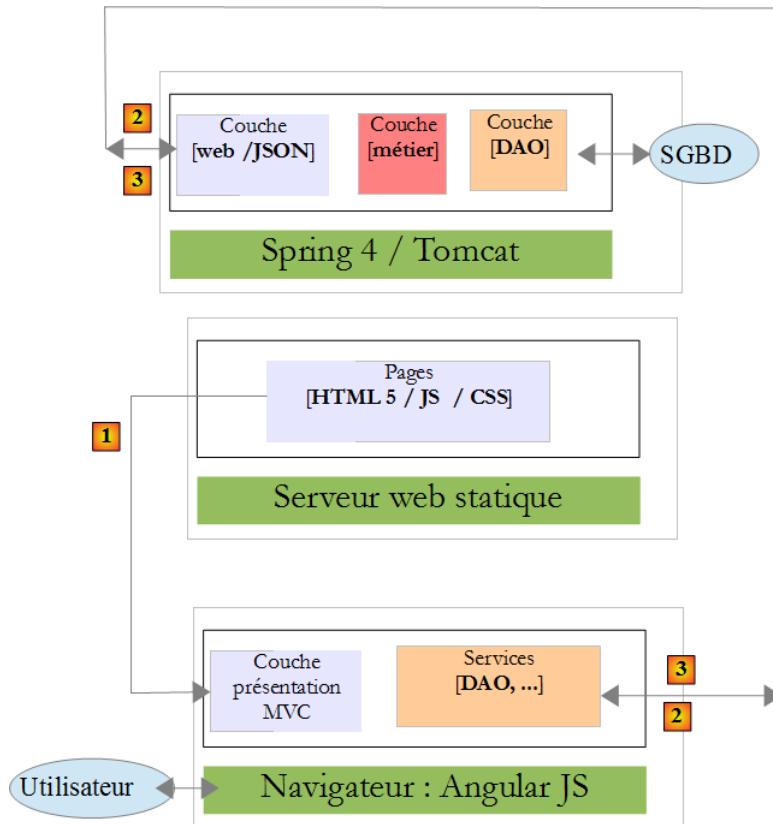
8

- en [8], la même vue que précédemment avec cette fois, un rendez-vous.

Au final, notre application s'adapte plutôt bien au smartphone. Cela pourrait être sûrement mieux mais cela reste utilisable.

5 Conclusion

Nous avons construit l'application client / serveur suivante :



Pour arriver à la version finale du code, nous avons du expliquer de nombreux points des frameworks AngularJS et Spring 4. Ce document peut donc être utilisé pour se former à l'utilisation de ces deux frameworks. Le paragraphe 1.3, page 7, explique où trouver les codes et comment les exploiter.

Nous avons montré que l'application client / serveur était utilisable dans divers environnements :

- comme une application web classique ;
- comme un binaire exécutable sur des émulateurs Android ;

Encore une fois, ce tutoriel n'est pas exhaustif quant à l'étude des deux frameworks. Pour Angular, il faudrait certainement présenter les outils de tests qui l'accompagnent. Les tests sont des étapes indispensables lors de l'écriture d'une application. Les outils qui gravitent autour d'Angular permettent de les automatiser et des inclure dans un processus d'intégration continue.

De ce travail, je retiendrai deux points :

- l'écriture du service web **Spring** a été moyennement compliquée. Dès le départ, je connaissais bien les concepts de Spring. Je n'ai rencontré de difficultés qu'avec la sécurisation du service web puis plus tard avec la gestion des entêtes HTTP CORS, deux domaines que je ne connaissais pas ;
- l'écriture du client **Angular** a été beaucoup plus complexe pour différentes raisons :
 - j'avais une connaissance insuffisante du langage Javascript et de ses possibilités ;
 - j'ai eu du mal à comprendre comment fonctionnait la programmation asynchrone au sein du navigateur. Je raisonnais comme sur un serveur où cet asynchronisme est obtenu avec l'utilisation simultanée de plusieurs threads. Dans le navigateur, il n'y a qu'un thread, et les tâches asynchrones sont traitées successivement et non pas en parallèle. Plus précisément, des tâches asynchrones peuvent s'exécuter en parallèle (requêtes HTTP multiples par exemple) mais les événements qu'elles produisent lorsqu'elles sont terminées, sont eux traités séquentiellement. Il n'y a donc pas d'exécution concurrente à gérer avec les nombreux problèmes qui vont avec ;

- Angular est un framework riche avec de nombreuses notions (MVC, directives, services, portée des modèles, ...). Son apprentissage est long ;
- Angular n'impose pas de méthode de développement. Ainsi pour arriver à un même résultat, on peut utiliser différentes architectures. C'est déroutant. Je suis plus à l'aise avec des frameworks fermés où tout le monde utilise les mêmes patrons de conception (design pattern). J'ai donc constamment cherché à reproduire les modèles de conception que j'utilise côté serveur. Je suis satisfait du résultat car je pense qu'il est reproductible. C'est ce que je cherchais. Mais je ne sais pas du tout si je me suis écarté ou non des « bonnes pratiques » d'Angular ;

Serge Tahé, juillet 2014.

6 Annexes

Nous présentons ici comment installer les outils utilisés dans ce document sur des machines Windows 7.

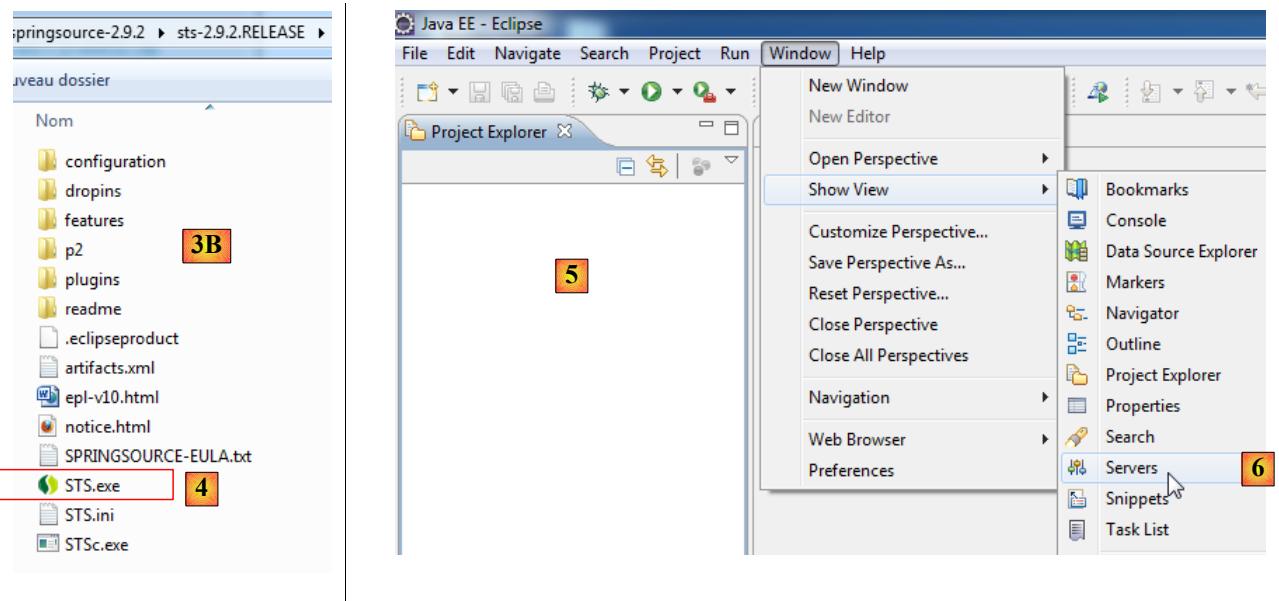
6.1 Installation de STS (Spring Tool Suite)

Nous allons installer **SpringSource Tool Suite** [<http://www.springsource.com/developer/sts>], un Eclipse pré-équipé avec de nombreux plugins liés au framework Spring et également avec une configuration Maven pré-installée.

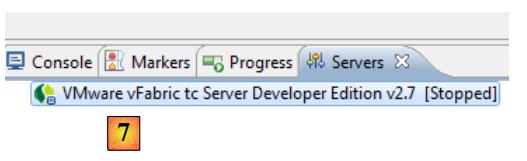
The screenshot shows the official website for SpringSource Tool Suite (STS). The URL in the address bar is www.springsource.com/developer/sts. A yellow box labeled '1' highlights the main title 'SpringSource Tool Suite — The Best Development Tool for Enterprise Java'. To the left is a sidebar with links like 'Developers Tools & Communities', 'Spring', 'Groovy & Grails', 'SpringSource Tool Suite' (which is highlighted with a red box labeled '2A'), 'tc Server Developer Edition', and 'WaveMaker'. The central content area features a description of STS's capabilities, a logo for 'springsource TOOLSUITE' (with two hammers forming a circle around it), and a testimonial from Charlie Babcock, Reporter at InformationWeek. At the bottom right is a large orange 'Download STS' button (labeled '2B') and a 'Contact Request: Support' link.

- aller sur le site de **SpringSource Tool Suite** (STS) [1], pour télécharger la version courante de STS [2A] [2B],

The screenshot consists of two parts. On the left, a file manager window titled 'Eclipse 3.7.2' shows two files for download: 'springsource-tool-suite-2.9.2.RELEASE-e3.7.2-win32-installer.exe' (labeled '2B') and 'springsource-tool-suite-2.9.2.RELEASE-e3.7.2-win32.zip'. On the right, a file manager window titled 'ams > devjava > Spring STS > springsource-2.9.2' shows a list of downloaded files: 'apache-maven-3.0.3' (labeled '3A'), 'spring-roo-1.2.1.RELEASE', 'sts-2.9.2.RELEASE', and 'vfabric-tc-server-developer-2.7.0.RELEASE'. The date '26/05,' is listed next to each file.



- le fichier téléchargé est un installateur qui crée l'arborescence de fichiers [3A] [3B]. En [4], on lance l'exécutable,
- en [5], la fenêtre de travail de l'IDE après avoir fermé la fenêtre de bienvenue. En [6], on fait afficher la fenêtre des serveurs d'applications,

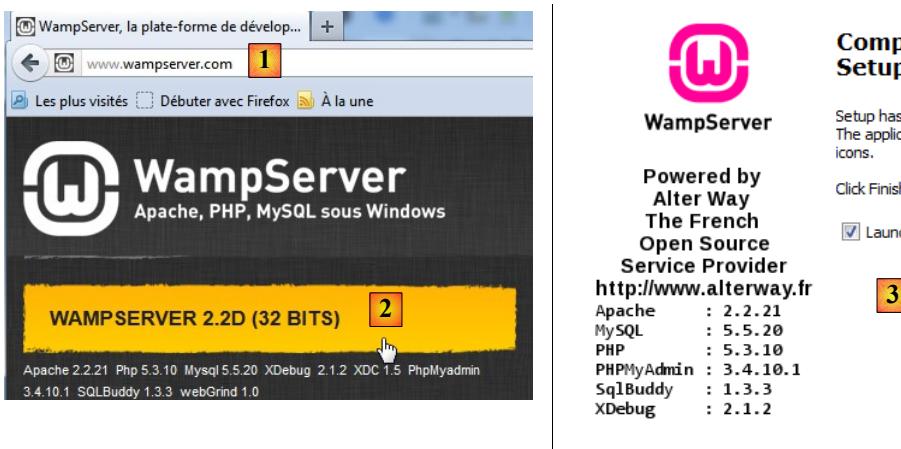


- en [7], la fenêtre des serveurs. Un serveur est enregistré. C'est un serveur VMware compatible Tomcat.

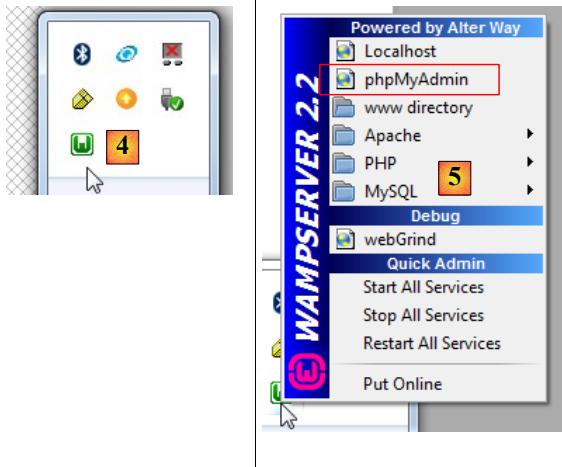
L'utilisation de STS dans le cadre de l'application est expliquée au paragraphe 1.3.2, page 9.

6.2 Installation de [WampServer]

[WampServer] est un ensemble de logiciels pour développer en PHP / MySQL / Apache sur une machine Windows. Nous l'utiliserons uniquement pour le SGBD MySQL.



- sur le site de [WampServer] [1], choisir la version qui convient [2],
- l'exécutable téléchargé est un installateur. Diverses informations sont demandées au cours de l'installation. Elles ne concernent pas MySQL. On peut donc les ignorer. La fenêtre [3] s'affiche à la fin de l'installation. On lance [WampServer],



- en [4], l'icône de [WampServer] s'installe dans la barre des tâches en bas et à droite de l'écran [4],
- lorsqu'on clique dessus, le menu [5] s'affiche. Il permet de gérer le serveur Apache et le SGBD MySQL. Pour gérer celui-ci, on utiliser l'option [PhpPmyAdmin],
- on obtient alors la fenêtre ci-dessous,

Nous donnerons peu de détails sur l'utilisation de [PhpMyAdmin]. Nous montrons au paragraphe 1.3.1, page 8, comment l'utiliser pour créer la base de données de l'application.

6.3 Installation de [Webstorm]

[WebStorm] (WS) est l'IDE de JetBrains pour développer des applications HTML / CSS / JS. Je l'ai trouvé parfait pour développer des applications Angular. Le site de téléchargement est [<http://www.jetbrains.com/webstorm/download/>]. C'est un IDE payant mais une version d'évaluation de 30 jours est téléchargeable. Il existe une version personnelle et une version étudiante peu onéreuses.

Son utilisation dans le cadre de l'application est décrite au paragraphe 1.3.3, page 11. Pour installer des bibliothèques JS au sein d'une application, WS utilise un outil appelé [bower]. Cet outil est un module de [node.js], un ensemble de bibliothèques JS. Par ailleurs, les bibliothèques JS sont cherchées sur un site Git, nécessitant un client Git sur le poste qui télécharge.

6.3.1 Installation de [node.js]

Le site de téléchargement de [node.js] est [<http://nodejs.org/>]. Téléchargez l'installateur puis exécutez-le. Il n'y a rien de plus à faire pour le moment.

6.3.2 Installation de l'outil [bower]

L'installation de l'outil [bower] qui va permettre le téléchargement des bibliothèques Javascript peut se faire de différentes façons. Nous allons la faire à partir de la console :

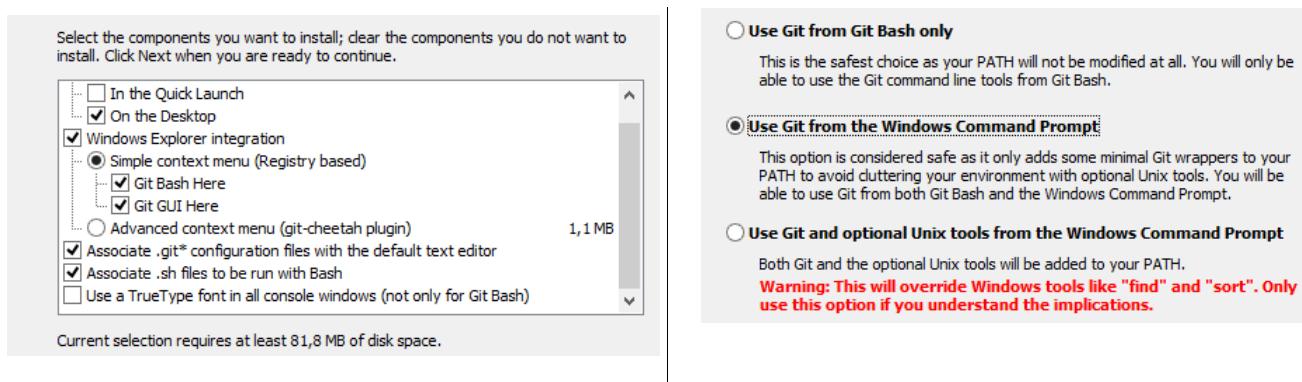
```
1. C:\Users\Serge Tahé>npm install -g bower
2. C:\Users\Serge Tahé\AppData\Roaming\npm\bower -> C:\Users\Serge
Tahé\AppData\Roaming\npm\node_modules\bower\bin\bower
3. bower@1.3.7 C:\Users\Serge Tahé\AppData\Roaming\npm\node_modules\bower
4.   ├── stringify-object@0.2.1
5.   ├── is-root@0.1.0
6.   └── junk@0.3.0
7. ...
8.   └── insight@0.3.1 (object-assign@0.1.2, async@0.2.10, lodash.debounce@2.4.1, req
9.   uest@2.27.0, configstore@0.2.3, inquirer@0.4.1)
10.  └── mout@0.9.1
11.  └── inquirer@0.5.1 (readline2@0.1.0, mute-stream@0.0.4, through@2.3.4, async@0.8
12. .0, lodash@2.4.1, cli-color@0.3.2)
```

- ligne 1 : la commande [node.js] qui installe le module [bower]. Pour que la commande marche, il faut que l'exécutable [npm] soit dans le PATH de la machine (voir paragraphe ci-après) ;

6.3.3 Installation de [Git]

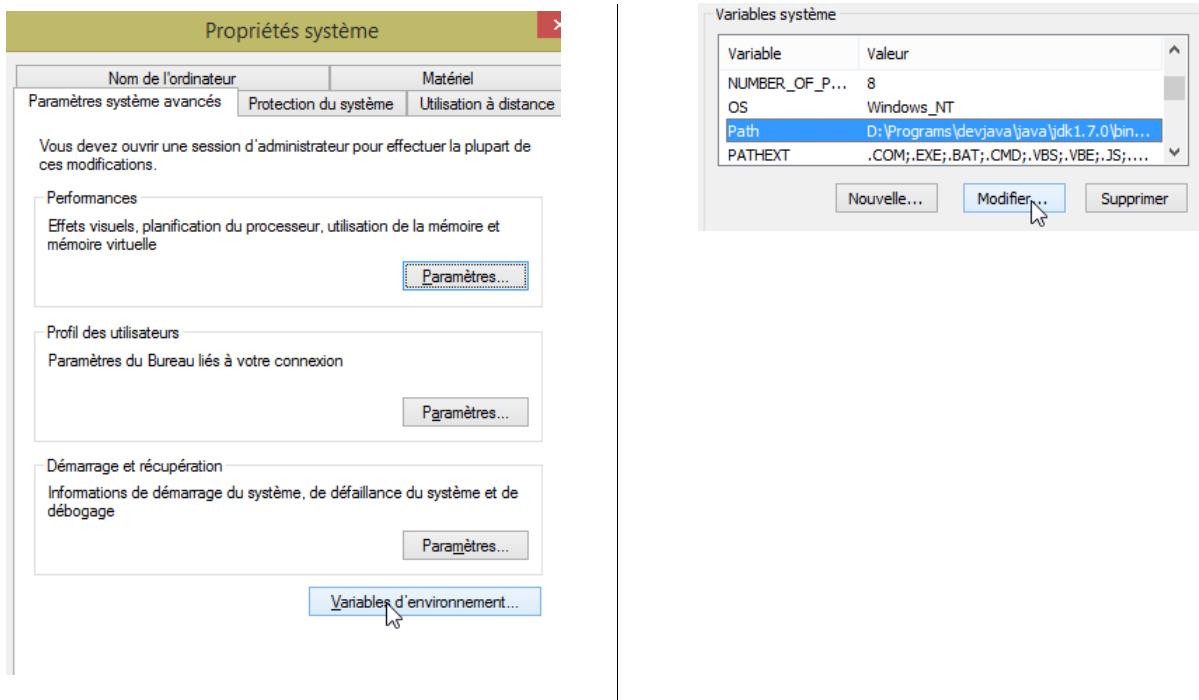
Git est un système de gestion de versions de logiciel. Il existe une version windows appelée [msysgit] et disponible à l'URL [<http://msysgit.github.io/>]. Nous n'allons pas utiliser [msysgit] pour gérer des versions de notre application mais simplement pour télécharger des bibliothèques JS qui se trouvent sur des sites de type [<https://github.com>] qui nécessitent un protocole d'accès spécial et qui est fourni par le client [msysgit]

L'assistant d'installation propose différentes étapes dont les suivantes :



Pour les autres étapes de l'installation, vous pouvez accepter les valeurs par défaut proposées.

Une fois, l'installation de Git terminée, vérifiez que l'exécutable est dans le PATH de votre machine : [Panneau de configuration / Système et sécurité / Système / Paramètres systèmes avancés] :



La variable PATH ressemble à ceci :

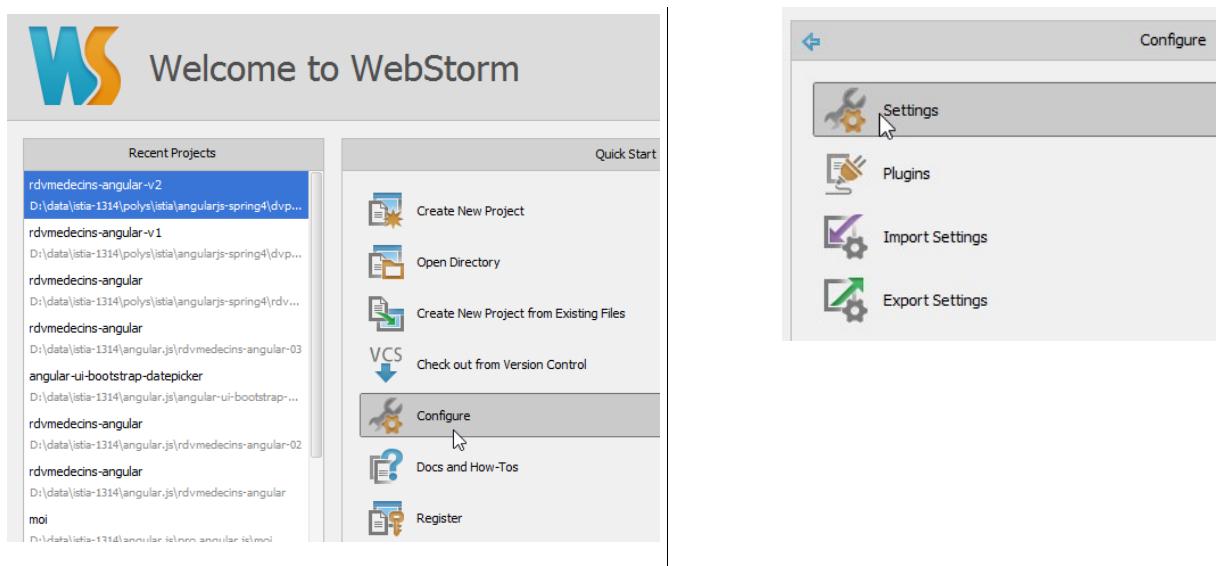
```
D:\Programs\devjava\java\jdk1.7.0\bin;D:\Programs\ActivePerl\Perl64\site\bin;D:\Programs\ActivePerl\Perl64\bin;D:\Programs\sgbd\OracleXE\app\oracle\product\11.2.0\client;D:\Programs\sgbd\OracleXE\app\oracle\product\11.2.0\client\bin;D:\Programs\sgbd\OracleXE\app\oracle\product\11.2.0\server\bin;...;D:\Programs\javascript\node.js\;D:\Programs\utilitaires\Git\cmd
```

Vérifiez que :

- le chemin du dossier d'installation de [node.js] est bien présent (ici D:\Programs\javascript\node.js) ;
- le chemin de l'exécutable du client Git est bien présent (ici D:\Programs\utilitaires\Git\cmd) ;

6.3.4 Configuration de [Webstorm]

Vérifions maintenant la configuration de [Webstorm]



Ci-dessus, sélectionnez l'option [1]. La liste des modules [node.js] déjà installés apparaît en [2]. Cette liste ne devrait contenir que la ligne [3] du module [bower] si vous avez suivi le processus d'installation précédent.

6.4 Installation d'un émulateur pour Android

Les émulateurs fournis avec le SDK d'Android sont lents ce qui décourage de les utiliser. L'entreprise [[Genymotion](#)] offre un émulateur beaucoup plus performant. Celui-ci est disponible à l'URL [<https://cloud.genymotion.com/page/launchpad/download/>] (février 2014).

Vous aurez à vous enregistrer pour obtenir une version à usage personnel. Téléchargez le produit [Genymotion] avec la machine virtuelle VirtualBox ;

Download ready-to-run Genymotion installer for Windows

This version includes Oracle VirtualBox 4.2.12 dependency, so that you don't need to download and install VirtualBox manually

Windows 32/64 bits (with VirtualBox) v2.1.1

Installez puis lancez [Genymotion]. Téléchargez ensuite une image pour une tablette ou un téléphone :

- en [1], ajoutez un terminal virtuel ;
- en [2], choisissez un ou plusieurs terminaux à installer. Vous pouvez affiner la liste affichée en précisant la version d'Android désirée [3] ainsi que le modèle de terminal [4] ;

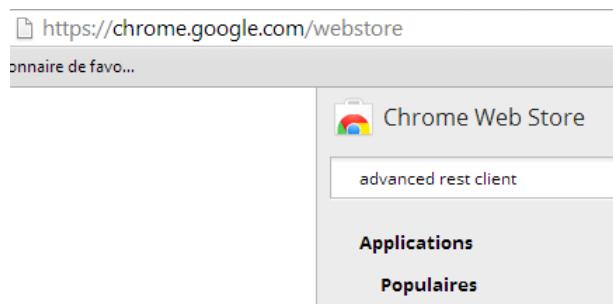


- une fois le téléchargement terminé, vous obtenez en [5] la liste des terminaux virtuels dont vous disposez pour tester vos applications Android ;

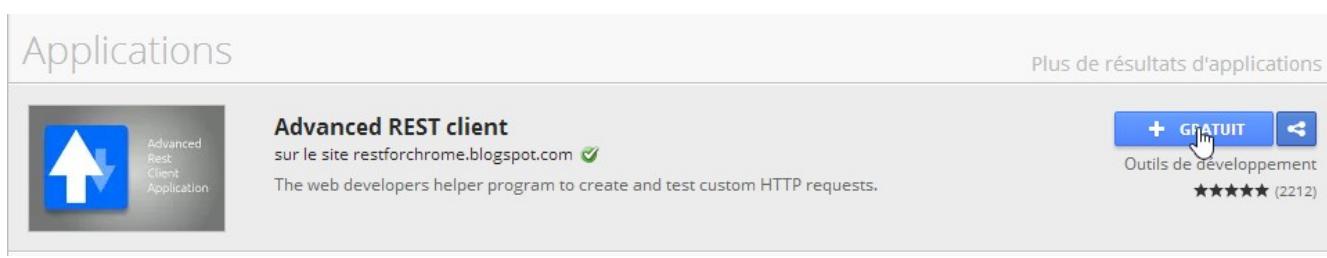
6.5 Installation du plugin Chrome [Advanced Rest Client]

Dans ce document, on utilise le navigateur **Chrome** de Google (<http://www.google.fr/intl/fr/chrome/browser/>). On lui ajoutera l'extension [Advanced Rest Client]. On pourra procéder ainsi :

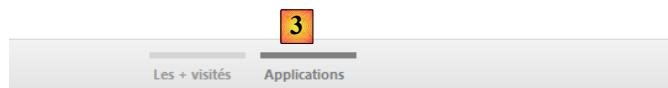
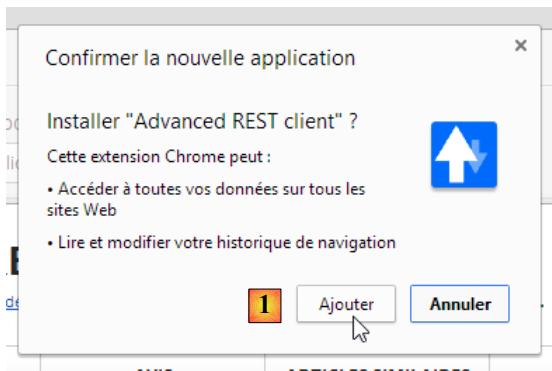
- aller sur le site de [Google Web store] (<https://chrome.google.com/webstore>) avec le navigateur Chrome ;
- chercher l'application [Advanced Rest Client] :



- l'application est alors disponible au téléchargement :



- pour l'obtenir, il vous faudra créer un compte Google. [Google Web Store] demande ensuite confirmation [1] :



- en [2], l'extension ajoutée est disponible dans l'option [Applications] [3]. Cette option est affichée sur chaque nouvel onglet que vous créez (CTRL-T) dans le navigateur.